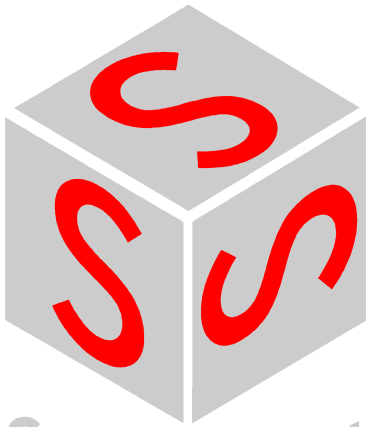
User Manual for PLC Programming with

CoDeSys 2.3



Copyright © 2003 by 3S - Smart Software Solutions GmbH
All rights reserved.

We have gone to great lengths to ensure this documentation is correct and complete. However, since it is not possible to produce an absolutely error-free text, please feel free to send us your hints and suggestions for improving it.

Trademark
Intel is a registered trademark and 80286, 80386, 80486, Pentium are trademarks of Intel Corporation.
Microsoft, MS and MS-DOS are registered trademarks, Windows is a trademark of Microsoft Corporation.

Publisher

3S - Smart Software Solutions GmbH
Fischerstraße 19
D-87435 Kempten
Tel. +49 831 5 40 31 - 0
Fax +49 831 5 40 31 - 50

Version 1.0
Last update 08.01.2003

Content

1 A Brief Introduction to CoDeSys 1-1

1.1 What is CoDeSys 1-1
1.2 Overview of CoDeSys Functions 1-1

2 What is What in CoDeSys 2-1

2.1 Project Components 2-1
2.2 Languages 2-9
2.2.1 Instruction List (IL) 2-9
2.2.2 Structured Text (ST) 2-11
2.2.3 Sequential Function Chart (SFC) 2-18
2.2.4 Function Block Diagram (FBD) 2-24
2.2.5 The Continuous Function Chart Editor (CFC) 2-25
2.2.6 Ladder Diagram (LD) 2-25
2.3 Debugging, Online Functions 2-27
2.4 The Standard 2-30

3 We Write a Little Program 3-1

3.1 Controlling a Traffic Signal Unit 3-1
3.2 Visualizing a Traffic Signal Unit 3-13

4 The Individual Components 4-1

4.1 The Main Window 4-1
4.2 Options 4-4
4.3 Managing Projects 4-23
4.4 Managing Objects in a Project 4-50
4.5 General Editing Functions 4-66
4.6 General Online Functions 4-73
4.7 Log 4-90
4.8 Window set up 4-92
4.9 Help when you need it 4-93

5 Editors in CoDeSys 5-1

5.1 Declaration Editor 5-2
5.2 The Text Editors 5-14
5.2.1 The Instruction List Editor 5-18
5.2.2 The Editor for Structured Text 5-19
5.3 The Graphic Editors 5-20
5.3.1 The Function Block Diagram Editor 5-22
5.3.2 The Ladder Editor 5-28

5.3.3 The Sequential Function Chart Editor 5-34
5.3.4 The Continuous Function Chart Editor (CFC) 5-44

6 The Resources 6-1

6.1 Global Variables 6-2
6.1.1 What are Global Variables 6-2
6.1.2 Variable Configuration 6-8
6.1.3 Document Frame 6-9
6.2 Library Manager 6-10
6.3 Log 6-12
6.4 PLC Browser 6-15
6.5 Parameter Manager 6-19
6.5.1 The OD Editor (Parameter Manager) 6-20
6.5.2 Program access to the OD (Object Dictionary, Para,eter Manager) data 6-23
6.6 PLC Configuration 6-23
6.6.1 Working in the PLC Configuration 6-25
6.6.2 General Settings 6-26
6.6.3 Application-specific parameter dialog (Custom Parameters) 6-27
6.6.4 Configuration of an I/O Module 6-28
6.6.5 Configuration of a channel 6-31
6.6.6 Configuration of Profibus Modules 6-32
6.6.7 Configuration of CAN modules 6-41
6.7 Task Configuration 6-47
6.8 Sampling Trace 6-55
6.9 Watch and Receipt Manager 6-61
6.10 Target Settings 6-64
6.10.1 Dialog Target Settings 6-65
6.11 Tools 6-66

7 Visualization 7-1

7.1 Create a new Visualization Object 7-2
7.1.1 Insert Visualization Elements 7-2
7.1.2 Positioning Visualization Elements 7-5
7.2 Configuration of a Visualization 7-8
7.2.1 Configure Visualization Elements 7-11
7.2.2 Configure Visualization Objects 7-31
7.2.3 Visualization in Online Mode 7-37
7.3 Visualizations in libraries 7-38
7.4 Web-Visualization 7-38
7.5 Target-Visualization 7-40

8 DDE Interface 8-1

8.1 DDE interface of the CoDeSys programming system 8-1
8.2 DDE communcation with the GatewayDDE Server 8-2

9 The CoDeSys ENI 9-7

9.1 Preconditions for Working with an ENI project data base 9-7
9.2 Working with the ENI project data base in CoDeSys 9-8
9.3 Object categories concerning the project data base 9-9

10 License Management in CoDeSys 10-11

10.1 Creating a licensed library in CoDeSys 10-11

11 APPENDIX 11-1

Appendix A: Use of Keyboard 11-1

Key Combinations 11-1

Appendix B: Data Types 11-5

Standard Data types 11-5
Defined Data Types 11-6

Appendix C: The IEC Operators and additional, norm-extending functions 11-13

Arithmetic Operators 11-13
Bitstring Operators 11-17
Bit-Shift Operators 11-19
Selection Operators 11-22
Comparison Operators 11-25
Address Operators 11-27
Calling Operator 11-28
Type Conversion Functions 11-28
Numeric Functions 11-34

Appendix D: Standard Library Elements 11-39

String functions 11-39
Bistable Function Blocks 11-43
Trigger 11-45
Counter 11-46
Timer 11-48

Appendix E: The UTIL.LIB Library 11-53

BCD conversion 11-53
Bit/byte functions 11-54
Mathematical help functions 11-55
Controllers 11-56
Signal generators 11-57
Function manipulators 11-59
Analogue value processing 11-61

Appendix F: Operands in CoDeSys 11-63

Constants in CoDeSys 11-63
Variables 11-65
Addresses 11-66
Functions 11-67

Appendix G: Command Line/Command File Commands 11-69

Command Line Commands 11-69
Command File (cmdfile) Commands 11-69

Appendix H: Siemens Import 11-77

Appendix I: Target Settings 11-85

Target system 'Intel 386 compatible' 11-86
Target system Motorola 68K 11-91
Target system Infineon C16x 11-92
Target systems Intel StrongARM und Power PC 11-93
Target system MIPS 11-94
Target system Hitachi SH 11-95
Target system 8051 compatible 11-96

Appendix J: Overview CoDeSys Operators and Library Elements 11-97

Appendix K: Compiler Errors and Warnings 11-103

Warnings 11-103
Compiler Errors 11-105

12 Index I

1 A Brief Introduction to CoDeSys

1.1 What is CoDeSys

CoDeSys is a complete development environment for your PLC. (**CoDeSys** stands for Controlled Developement System).

CoDeSys puts a simple approach to the powerful IEC language at the disposal of the PLC programmer. Use of the editors and debugging functions is based upon the proven development program environments of advanced programming languages (such as Visual C++).

1.2 Overview of CoDeSys Functions

How is a project structured?

A project is put into a file named after the project. The first POU (Program Organization Unit) created in a new project will automatically be named **PLC_PRG**. The process begins here (in compliance with the main function in a C program), and other POUs can be accessed from the same point (programs, function blocks and functions).

Once you have defined a Task Configuration, it is no longer necessary to create a program named PLC_PRG. You will find more about this in the Task Configuration chapter.

There are different kinds of objects in a project: POUs, data types, display elements (visualizations) and resources.

The Object Organizer contains a list of all the objects in your project.

How do I set up my project?

First you should configure your PLC in order to check the accuracy of the addresses used in the project.

Then you can create the POUs needed to solve your problem.

Now you can program the POUs you need in the desired languages.

Once the programming is complete, you can compile the project and remove errors should there be any.

How can I test my project?

Once all errors have been removed, activate the simulation, log in to the simulated PLC and "load" your project in the PLC. Now you are in Online mode.

Now open the window with your PLC Configuration and test your project for correct sequence. To do this, enter input variables manually and observe whether outputs are as expected. You can also observe the value sequence of the local variables in the POUs. In the Watch and Receipt Manager you can configure data records whose values you wish to examine.

Debugging

In case of a programming error you can set breakpoints. If the process stops at such a breakpoint, you can examine the values of all project variables at this point in time. By working through sequentially (single step) you can check the logical correctness of your program.

Additional Online Functions

Further debugging functions:

You can **set** program variables and inputs and outputs **at certain values**.

You can use the **flow control** to check which program lines have been run.

A **Log** records operations, user actions and internal processes during an online session in a chronological order.

The **Sampling Trace** allows you to trace and display the actual course of variables over an extended period of time.

Optional a **PLC Browser** is available to request certain information from the PLC.

Once the project has been set up and tested, it can be **loaded down to the hardware** and tested as well. The same online functions as you used with the simulation will be available.

*Additional **CoDeSys** Features*

The entire project can be **documented** or **exported** to a text file at any time. Also it can be **translated** into another language.

CoDeSys has a symbolic and a DDE interface. A gateway server together with an OPC server and DDE server are parts of the **CoDeSys** standard installation.

Network variables and an **Object Dictionary** can be used for data exchange with other PLCs within a network.

Summary

CoDeSys is a complete development tool used to program your PLC which will save you a measurable amount of time setting up your applications.

2 What is What in CoDeSys

This chapter contains a list of the most important concepts to make starting easier.

2.1 Project Components

Project

A project contains all of the objects in a PLC program. A project is saved in a file named after the project. The following objects are included in a project:

POUs (Program Organization Units), data types, visualizations, resources, and libraries.

POU (Program Organization Unit)

Functions, function blocks, and programs are POUs which can be supplemented by actions.

Each POU consists of a declaration part and a body. The body is written in one of the IEC programming languages which include IL, ST, SFC, FBD, LD or CFC.

CoDeSys supports all IEC standard POUs. If you want to use these POUs in your project, you must include the library standard.lib in your project.

POUs can call up other POUs. However, recursions are not allowed.

Function

A function is a POU, which yields exactly one data element (which can consist of several elements, such as fields or structures) when it is processed, and whose call in textual languages can occur as an operator in expressions.

When declaring a function do not forget that the function must receive a type. This means, after the function name, you must enter a colon followed by a type.

A correct function declaration can look like this example:

```
FUNCTION Fct:        INT
```

In addition, a result must be assigned to the function. That means that function name is used as an output variable.

A function declaration begins with the keyword FUNCTION.

Example in IL of a function that takes three input variables and returns the product of the first two divided by the third:

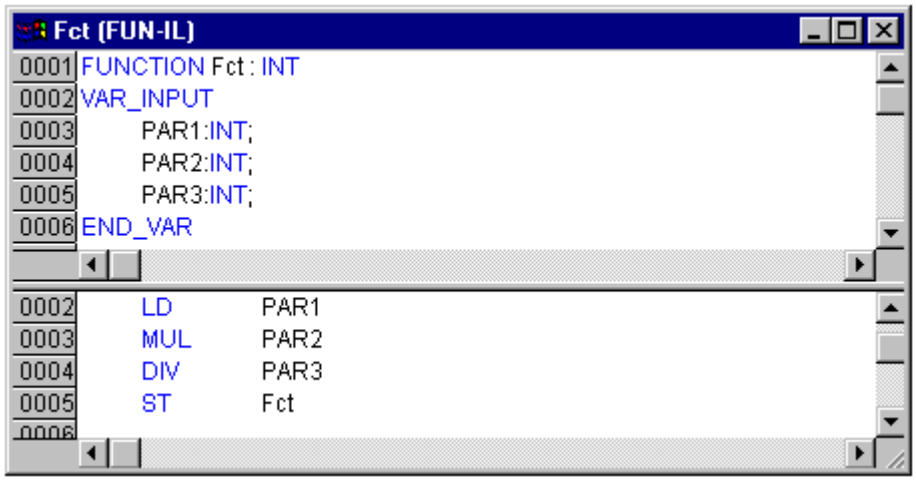


Image 2.1: Example of a function in IL

The call of a function in ST can appear as an operand in expressions.

Functions do not have any internal conditions. That means that calling up a function with the same argument (input parameters) always produces the same value (output).

Examples for calling up the function described above:

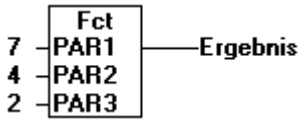
in IL:

```
LD   7
Fct  2,4
ST   Result
```

in ST:

```
Result := Fct(7, 2, 4);
```

in FBD:



In SFC a function call can only take place within a step or a transition.

Note:

If you define a function in your project with the name **CheckBounds**, you can use it to check range overflows in your project! The name of the function is defined and may have only this identifier (see Appendix B, Data types) .

If you define functions in your project with the names **CheckDivByte**, **CheckDivWord**, **CheckDivDWord** and **CheckDivReal**, you can use them to check the value of the divisor if you use the operator DIV, for example to avoid a division by 0 (see Appendix Appendix C: , IEC operators).

If you define functions with the names **CheckRangeSigned** and **CheckRangeUnsigned**, then range exceeding of variables declared wit subrange types (see Appendix Appendix B: , Data types) can be intercepted.

All these check function names are reserved for the described usage.

Function Block

A function block is a POU which provides one or more values during the procedure. As opposed to a function, a function block provides no return value.

A function block declaration begins with the keyword FUNCTION_BLOCK.

Example in IL of a function block with two input variables and two output variables. One output is the product of the two inputs, the other a comparison for equality:

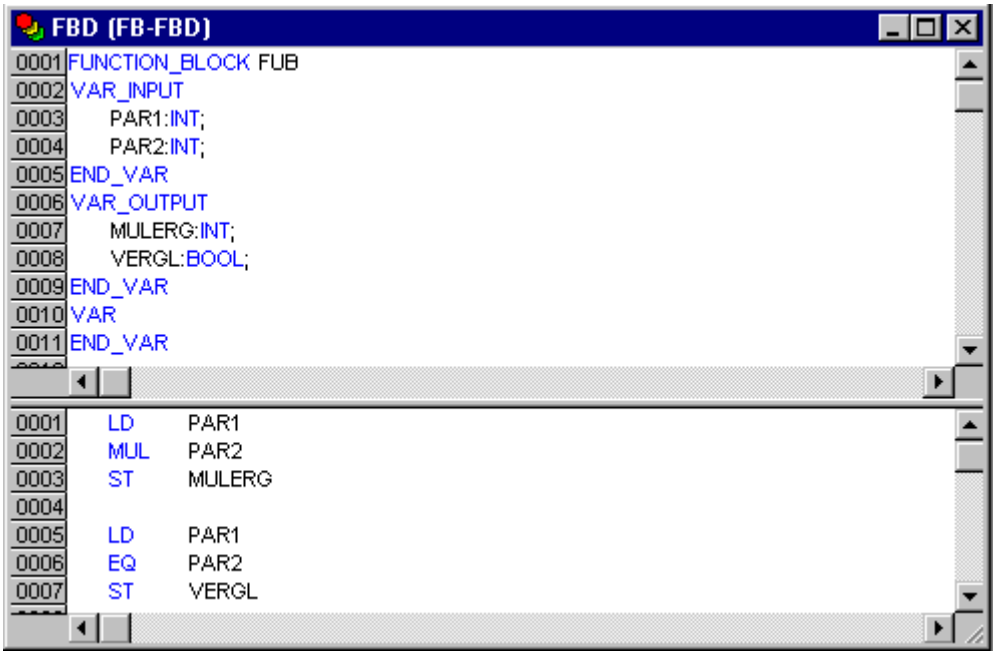


Image 2.2: Function Block

Function Block Instances

Reproductions or *instances* (copies) of a function block can be created.

Each instance possesses its own identifier (the Instance name), and a data structure which contains its inputs, outputs, and internal variables. Instances are declared locally or globally as variables, whereas the name of the function block is indicated as the type of an identifier.

Example of an instance with the name INSTANCE of the FUB function block:

```
INSTANCE: FUB;
```

Function blocks are always called through the instances described above.

Only the input and output parameters can be accessed from outside of an function block instance, not its internal variables.

Example for accessing an input variable:

The function block FB has an input variable in1 of the type INT.

```
PROGRAM prog
VAR
   inst1:fb;
END_VAR
LD      17
ST      inst1.in1
```

```
CAL      inst1
END_PROGRAM
```

The declaration parts of function blocks and programs can contain instance declarations. Instance declarations are not permitted in functions.

Access to a function block instance is limited to the POU in which it was declared unless it was declared globally.

The instance name of a function block instance can be used as the input for a function or a function block.

Note: All values are retained after processing a function block until the next it is processed. Therefore, function block calls with the same arguments do not always return the same output values!

Note: If there at least one of the function block variables is a retain variable, the total instance is stored in the retain area.

Calling a function block

The input and output variables of a function block can be accessed from another POU by setting up an instance of the function block and specifying the desired variable using the following syntax:

<Instance name>.<Variable name>

If you would like to set the input parameters when you open the function block, you can do this in the text languages IL and ST by assigning values to the parameters after the instance name of the function block in parentheses (this assignment takes place using ":=" just as with the initialization of variables at the declaration position).

Please regard, that the InOutVariables (VAR_IN_OUT) of a function block are handed over as pointers. For this reason in a call of a function block no constants can be assigned to VAR_IN_OUTs and there is no read or write access from outside to them.

```
VAR
   inst:fubo;
   var:int;
END_VAR
var1:=2;
inst(instout1:=var1^);
```

not allowed in this case: inst(instout1:=2); or inst.inout1:=2;

Examples for calling function block FUB described above:

The multiplication result is saved in the variable ERG, and the result of the comparison is saved in QUAD. An instance of FUB with the name INSTANCE is declared.

In IL the function block is called as shown in the following image:

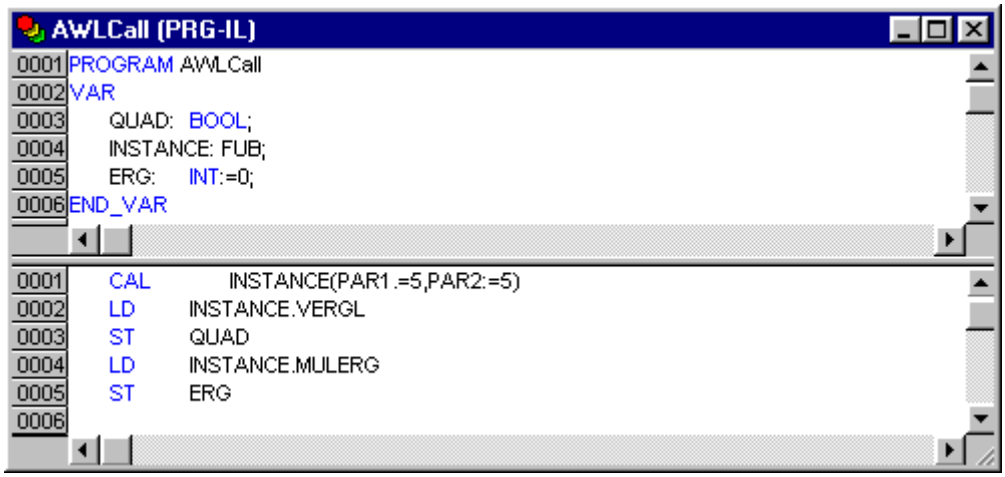


Image 2.3: Function Block Call in IL

In the example below the call is shown in ST. The declaration part is the same as with IL:

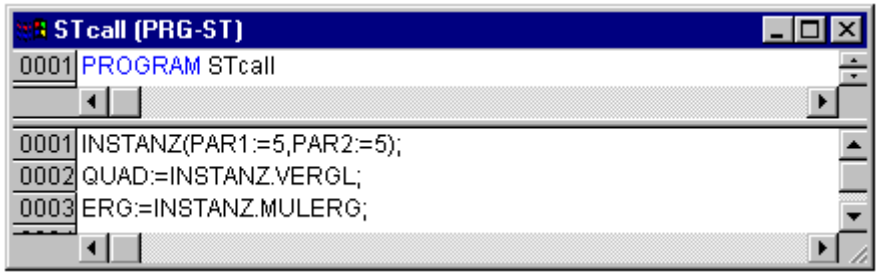


Image 2.4: Function Block Call in ST

In FBD the instance of a function block is called as shown in the following image (declaration part the same as with IL):

EINFÜGENGRAFIKFORMATVERBINDEN

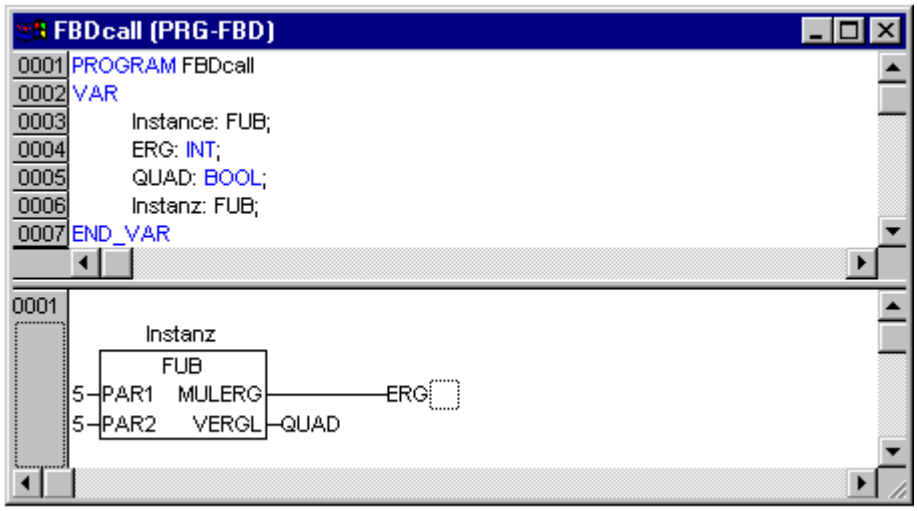


Image 2.5: Function Block Call in FBD

In SFC function block calls can only take place in steps.

Program

A program is a POU which returns several values during operation. Programs are recognized globally throughout the project. All values are retained from the last time the program was run until the next.

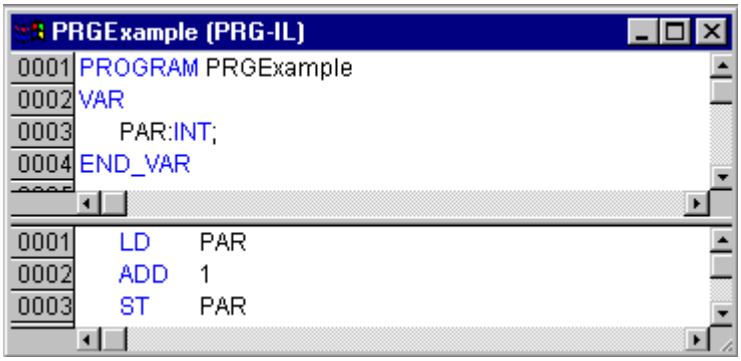


Image 2.6: Example of a program

Programs can be called. A program call in a function is not allowed. There are also no instances of programs.

If a POU calls a program, and if thereby values of the program are changed, then these changes are retained the next time the program is called, even if the program has been called from within another POU.

This is different from calling a function block. There only the values in the given instance of a function block are changed.

These changes therefore play a role only when the same instance is called.

A program declaration begins with the keyword PROGRAM and ends with END_PROGRAM.

Examples of calls of the program described above:

In IL:

```
CAL     PRGExample
LD      PRGexample.PAR
STERG
in ST:
PRGExample;
Erg := PRGexample.PAR;
```

In FBD:

Example for a possible call sequence for PLC_PRG:

```
LD      0
STPRGexample.PAR     (*Default setting for PAR is 0*)
CAL     IL call  (*ERG in IL call results in 1*)
CAL     ST call (*ERG in ST call results in 2*)
CAL     FBD call        (*ERG in FBD call results in 3*)
```

If the variable PAR from the program PRGexample is initialized by a main program with 0, and then one after the other programs are called with above named program

calls, then the ERG result in the programs will have the values 1, 2, and 3. If one exchanges the sequence of the calls, then the values of the given result parameters also change in a corresponding fashion.

PLC_PRG

The PLC_PRG is a special predefined POU. Each project must contain this special program. This POU is called exactly once per control cycle.

The first time the **'Project' 'Object Add'** command is used after a new project has been created, the default entry in the POU dialog box will be a POU named PLC_PRG of the program type. You should not change this default setting!

If tasks have been defined, then the project may not contain any PLC_PRG, since in this case the procedure sequence depends upon the task assignment.

Attention: Do not delete or rename the POU PLC_PRG (assuming you are not using a Task Configuration, see Chapter 6.7, Resources)! PLC_PRG is generally the main program in a single task program.

Action

Actions can be defined to function blocks and programmes. (For this purpose mark the function block or program in the Object Organizer and perform the command 'Project' 'Add action'. The new action will be inserted below the corresponding POU in the Object Organizer.) The action represents a further implementation which can be entirely created in another language as the ”normal” implementation. Each action is given a name.

An action works with the data from the function block or programme which it belongs to. The action uses the same input/output variables and local variables as the ”normal” implementation uses.

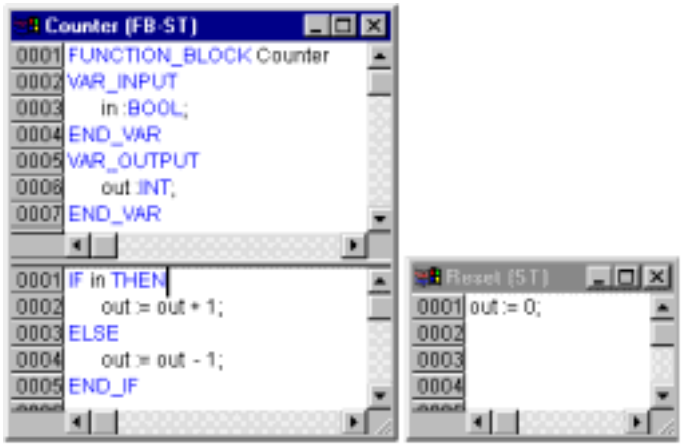


Image 2.7:Example for an action of a function block

In the example given, calling up the function block Counter increases or decreases the output variable ”out”, depending on the value of the input variable ”in”. Calling up the action Reset of the function block sets the output variable to zero. The same variable ”out” is written in both cases.

An action is called up with <Program_name>.<Action_name> or <Instance_name>.<Action_name>. If it is required to call up the action within its

own block, one just uses the name of the action in the text editors and in the graphic form the function block call up without instance information.

Examples for calls of the above described action from another POU:

Declaration for all examples:

```
PROGRAM PLC_PRG
VAR
    Inst : Counter;
END_VAR
```

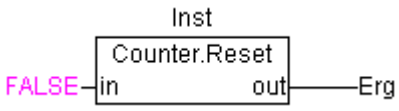
Call in **IL**:

```
CAL     Inst.Reset(In := FALSE)
LD      Inst.out
ST      ERG
```

Call in **ST**:

```
Inst.Reset(In := FALSE);
Erg := Inst.out;
```

Call in **FBD**:



Note: Actions play an important role in blocks in sequential function charts, see Chapter 2.2.3, Sequential Function Chart.

The IEC standard does not recognise actions other than actions of the sequential function chart.The IEC standard does not recognise actions other than actions of the sequential function chart.

Resources

You need the resources for configuring and organizing your project and for tracing variable values:

- Global Variables which can be used throughout the project or network
- Library manager for adding libraries to the project
- Log for recording the actions during an online session
- Object dictionary for data exchange with other controllers in a network
- PLC-Browser as controller monitor
- PLC Configuration for configuring your hardware
- Task Configuration for guiding your program through tasks
- Sampling Trace for graphic display of variable values
- Watch and Receipt Manager for displaying variable values and setting default variable values
- Target system settings for selection and if necessary final configuration of the target system
- Tools – availability depending on target – for calling external tool programs from within CoDeSys

See Chapter 6, 'The Resources'.

Libraries

You can include in your project a series of libraries whose POUs, data types, and global variables you can use just like user-defined variables. The libraries standard.lib and util.lib are standard parts of the program and are always at your disposal.

See Chapter 6.2, 'Library Manager'.

Data types

Along with the standard data types the user can define his own data types. Structures, enumeration types and references can be created.

See Appendix Appendix B: , 'Data Types'.

Visualization

CoDeSys provides visualizations so that you can display your project variables. You can plot geometric elements off-line with the help of the visualization. In Online mode, these can then change their form/color/text output in response to specified variable values.

See chapter 8, 'Visualization'.

2.2 Languages

CoDeSys supports all languages described by the standard IEC-61131:

Textual Languages:
- Instruction List (IL)InstructionList.IL.>Main
- Structured Text (ST)
- StructuredText.ST.>Main

Grafic Languages:
- Sequential Function Chart (SFC)SequentialFunctionChart.SFC.>Main
- Function Block Diagram (FBD)FunctionBlockDiagram.FBD.>Main
- The Continuous Function Chart Editor (CFC)TheContinuousFunctionChartEditor.CFC.1>Main
- Ladder Diagram (LD)LadderDiagram.LD.>Main

2.2.1 Instruction List (IL)

An instruction list (IL) consists of a series of instructions. Each instruction begins in a new line and contains an operator and, depending on the type of operation, one or more operands separated by commas.

In front of an instruction there can be an identification *mark* (label) followed by a colon (:).

A comment must be the last element in a line. Empty lines can be inserted between instructions.

Example:

```
LD      17
ST      lint            (* comment *)
GE      5
JMPC    next
LD      idword
EQ      istruct.sdword
STN     test
next:
```

Modifiers and operators in IL

In the IL language the following operators and modifiers can be used.

Modifiers:

- C with JMP, CAL, RET: The instruction is only then executed if the result of the preceding expression is TRUE.
- N with JMPC, CALC, RETC: The instruction is only then executed if the result of the preceding expression is FALSE.
- N otherwise: Negation of the operand (not of the accumulator)

Below you find a table of all operators in IL with their possible modifiers and the relevant meaning:

Operator	Modifiers	Meaning
LD	N	Make current result equal to the operand
ST	N	Save current result at the position of the operand
S		Then put the Boolean operand exactly at TRUE if the current result is TRUE
R		Then put the Boolean operand exactly at FALSE if the current result is TRUE
AND	N,(	Bitwise AND
OR	N,(	Bitwise OR
XOR	N,(	Bitwise exclusive OR
ADD	(	Addition
SUB	(	Subtraction
MUL	(	Multiplication
DIV	(	Division
GT	(	>
GE	(	>=
EQ	(	=
NE	(	<>
LE	(	<=
LT	(	<

JMP	CN	Jump to the label
CAL	CN	Call programor function block or
RET	CN	Leave POU and return to caller.
)		Evaluate deferred operation

You find a list of all IEC operators in the appendix.

Example of an IL program while using some modifiers:

```
LD      TRUE    (*load TRUE in the accumulator*)
ANDN    BOOL1   (*execute AND with the negated value of the BOOL1
                 variable*)
JMPC    mark    (*if the result was TRUE, then jump to the label "mark"*)

LDN     BOOL2   (*save the negated value of *)
ST      ERG     (*BOOL2 in ERG*)
label:
LD      BOOL2   (*save the value of *)
ST      ERG     *BOOL2 in ERG*)
```

It is also possible in IL to put parentheses after an operation. The value of the parenthesis is then considered as an operand.

For example:

```
LD  2
MUL 2
ADD 3
ST  ERG
```

Here is the value of Erg 7. However, if one puts parentheses:

```
LD  2
MUL (2
ADD 3
)
ST  ERG
```

Here the resulting value for Erg is 10, the operation MUL is only then evaluated if you come to ")"; as operand for MUL 5 is then calculated.

2.2.2 Structured Text (ST)

The Structured Text consists of a series of instructions which, as determined in high level languages, ("IF..THEN..ELSE) or in loops (WHILE..DO) can be executed.

Example:

```
IF value < 7 THEN
        WHILE       value < 8 DO
```

```
        value:=value+1;
        END_WHILE;
END_IF;
```

Expressions

An *expression* is a construction which returns a value after its evaluation.

Expressions are composed of operators and operands. An operand can be a constant, a variable, a function call, or another expression.

Valuation of expressions

The evaluation of expression takes place by means of processing the operators according to certain *binding rules*. The operator with the strongest binding is processed first, then the operator with the next strongest binding, etc., until all operators have been processed.

Operators with equal binding strength are processed from left to right.

Below you find a table of the ST operators in the order of their binding strength:

Operation	**Symbol**	**Binding strength**
Put in parentheses	(expression)	Strongest binding
Function call	Function name (parameter list)	
Exponentiation	EXPT	
Negate Building of complements	- NOT	
Multiply Divide Modulo	* / MOD	
Add Subtract	+ -	
Compare	<,>,<=,>=	
Equal to Not equal to	= <>	
Boolean AND	AND	
Boolean XOR	XOR	
Boolean OR	OR	Weakest binding

There are the following instructions in ST, arranged in a table together with example:

Instruction type	**Example**
Assignment	A:=B; CV := CV + 1; C:=SIN(X);

Calling a function block and use of the FB output	CMD_TMR(IN := %IX5, PT := 300); A:=CMD_TMR.Q
RETURN	RETURN;

IF	D:=B*B;
	IF D<0.0 THEN
	  C:=A;
	ELSIF D=0.0 THEN
	  C:=B;
	ELSE
	  C:=D;
	END_IF;

CASE	CASE INT1 OF
	1: BOOL1 := TRUE;
	2: BOOL2 := TRUE;
	ELSE
	  BOOL1 := FALSE;
	  BOOL2 := FALSE;
	END_CASE;

FOR	J:=101;
	FOR I:=1 TO 100 BY 2 DO
	  IF ARR[I] = 70 THEN
	  J:=I;
	  EXIT;
	  END_IF;
	END_FOR;

WHILE	J:=1; WHILE J<= 100 AND ARR[J] <> 70 DO J:=J+2; END_WHILE;
REPEAT	J:=-1; REPEAT J:=J+2; UNTIL J= 101 OR ARR[J] = 70 END_REPEAT;
EXIT	EXIT;
Empty instruction	;

Instruction in Structured Text

The name already indicates, the Structured Text is designed for structure programming, i.e. ST offers predetermined structures for certain often used constructs such as loops for programming.

This offers the advantages of low error probability and increased clarity of the program.

For example, let us compare two equally significant program sequences in IL and ST:

A loop for calculating powers of two in **IL**:

```
Loop:
LD      Counter
JMPC    end
LD      Var1
MUL     2
ST      Var1
LD      Counter
SUB     1
ST      Counter
JMP     Loop
End:
LD      Var1
ST      ERG
```

The same loop programmed in ST would produce:

```
WHILE counter<>0 DO
   Var1:=Var1*2;
   Counter:=counter-1;
END_WHILE
Erg:=Var1;
```

You can see, the loop in ST is not only faster to program, but is also significantly easier to read, especially in view of the convoluted loops in larger constructs.

The different structures in ST have the following significance:

Assignment operator

On the left side of an assignment there is an operand (variable, address) to which is assigned the value of the expression on the right side with the assignment operator :=

Example:

```
Var1 := Var2 * 10;
```

After completion of this line Var1 has the tenfold value of Var2.

Calling function blocks in ST

A function block is called in ST by writing the name of the instance of the function block and then assigning the values of the parameters in parentheses. In the following example a timer is called with assignments for the parameters IN and PT. Then the result variable Q is assigned to the variable A.

The result variable, as in IL, is addressed with the name of the function block, a following point, and the name of the variable:

```
CMD_TMR(IN := %IX5, PT := 300);
A:=CMD_TMR.Q
```

RETURN instruction

The RETURN instruction can be used to leave a POU, for example depending on a condition

IF instruction

With the IF instruction you can check a condition and, depending upon this condition, execute instructions.

Syntax:

IF <Boolean_expression1> **THEN**
<IF_instructions>
{ELSIF <Boolean_expression2> **THEN**
<ELSIF_instructions1>
.
.
ELSIF <Boolean_expression n> **THEN**
<ELSIF_instructions n-1>
ELSE
<ELSE_instructions>**}**
END_IF;

The part in braces {} is optional.

If the <Boolean_expression1> returns TRUE, then only the <IF_Instructions> are executed and none of the other instructions.

Otherwise the Boolean expressions, beginning with <Boolean_expression2>, are evaluated one after the other until one of the expressions returns TRUE. Then only those instructions after this Boolean expression and before the next ELSE or ELSIF are evaluated.

If none of the Boolean expressions produce TRUE, then only the <ELSE_instructions> are evaluated.

Example:

```
IF temp<17
THEN    heating_on := TRUE;
ELSE    heating_on := FALSE;
END_IF;
```

Here the heating is turned on when the temperature sinks below 17 degrees. Otherwise it remains off.

CASE instruction

With the CASE instructions one can combine several conditioned instructions with the same condition variable in one construct.

Syntax:

CASE <Var1> **OF**

<Value1>**:** <Instruction 1>
<Value2>**:** <Instruction 2>
<Value3, Value4, Value5>: <Instruction 3>
<Value6 .. Value10>: <Instruction 4>
...
<Value n>**:** <Instruction n>
ELSE <ELSE instruction>
END_CASE;

A CASE instruction is processed according to the following model:

- If the variable in <Var1> has the value <Value i>, then the instruction <Instruction i> is executed.
- If <Var 1> has none of the indicated values, then the <ELSE Instruction> is executed.
- If the same instruction is to be executed for several values of the variables, then one can write these values one after the other separated by commas, and thus condition the common execution.
- If the same instruction is to be executed for a value range of a variable, one can write the initial value and the end value separated by two dots one after the other. So you can condition the common condition.

Example:

```
CASE INT1 OF
1, 5:       BOOL1 := TRUE;
   BOOL3 := FALSE;
2:  BOOL2 := FALSE;
   BOOL3 := TRUE;
10..20: BOOL1 := TRUE;
   BOOL3:= TRUE;
ELSE
   BOOL1 := NOT BOOL1;
   BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

FOR loop

With the FOR loop one can program repeated processes.

Syntax:

```
INT_Var :INT;

FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <Step size>}
   DO
  <Instructions>
END_FOR;
```

The part in braces {} is optional.

The <Instructions> are executed as long as the counter <INT_Var> is not greater than the <END_VALUE>. This is checked before executing the <Instructions> so that the <instructions> are never executed if <INIT_VALUE> is greater than <END_VALUE>.

When <Instructions> are executed, <INT_Var> is always increased by <Step size>. The step size can have any integer value. If it is missing, then it is set to 1. The loop must also end since <INT_Var> only becomes greater.

Example:

```
FOR Counter:=1 TO 5 BY 1 DO
Var1:=Var1*2;
END_FOR;
Erg:=Var1;
```

Let us assume that the default setting for Var1 is the value 1. Then it will have the value 32 after the FOR loop.

Note: <END_VALUE> must not be equal to the limit value of the counter <INT_VAR>. For example: If the variable Counter is of type SINT and if <END_VALUE> is 127, you will get an endless loop.

WHILE loop

The WHILE loop can be used like the FOR loop with the difference that the break-off condition can be any Boolean expression. This means you indicate a condition which, when it is fulfilled, the loop will be executed.

Syntax:

```
WHILE <Boolean expression>
  <Instructions>
END_WHILE;
```

The <Instructions> are repeatedly executed as long as the <Boolean_expression> returns TRUE. If the <Boolean_expression> is already FALSE at the first evaluation, then the <Instructions> are never executed. If <Boolean_expression> never assumes the value FALSE, then the <Instructions> are repeated endlessly which causes a relative time delay.

Note: The programmer must make sure that no endless loop is caused. He does this by changing the condition in the instruction part of the loop, for example, by counting up or down one counter.

Example:

```
WHILE counter<>0 DO
    Var1 := Var1*2;
    Counter := Counter-1;
END_WHILE
```

The WHILE and REPEAT loops are, in a certain sense, more powerful than the FOR loop since one doesn't need to know the number of cycles before executing the loop. In some cases one will, therefore, only be able to work with these two loop types. If, however, the number of the loop cycles is clear, then a FOR loop is preferable since it allows no endless loops.

REPEAT loop

The REPEAT loop is different from the WHILE loop because the break-off condition is checked only after the loop has been executed. This means that the loop will run through at least once, regardless of the wording of the break-off condition.

Syntax:

```
REPEAT
  <Instructions>
UNTIL <Boolean expression>
END_REPEAT;
```

The <Instructions> are carried out until the <Boolean expression> returns TRUE.

If <Boolean expression> is produced already at the first TRUE evaluation, then <Instructions> are executed only once. If <Boolean_expression> never assumes the value TRUE, then the <Instructions> are repeated endlessly which causes a relative time delay.

Note: The programmer must make sure that no endless loop is caused. He does this by changing the condition in the instruction part of the loop, for example by counting up or down one counter.

Example:

```
REPEAT
   Var1 := Var1*2;
   Counter := Counter-1;
UNTIL
   Counter=0
END_REPEAT;
```

EXIT instruction

If the EXIT instruction appears in a FOR, WHILE, or REPEAT loop, then the innermost loop is ended, regardless of the break-off condition.

2.2.3 Sequential Function Chart (SFC)

The Sequential Function Chart is a graphically oriented language which makes it possible to describe the chronological order of different actions within a program. For this the actions are assigned to step elements and the sequence of processing is controlled by transition elements.

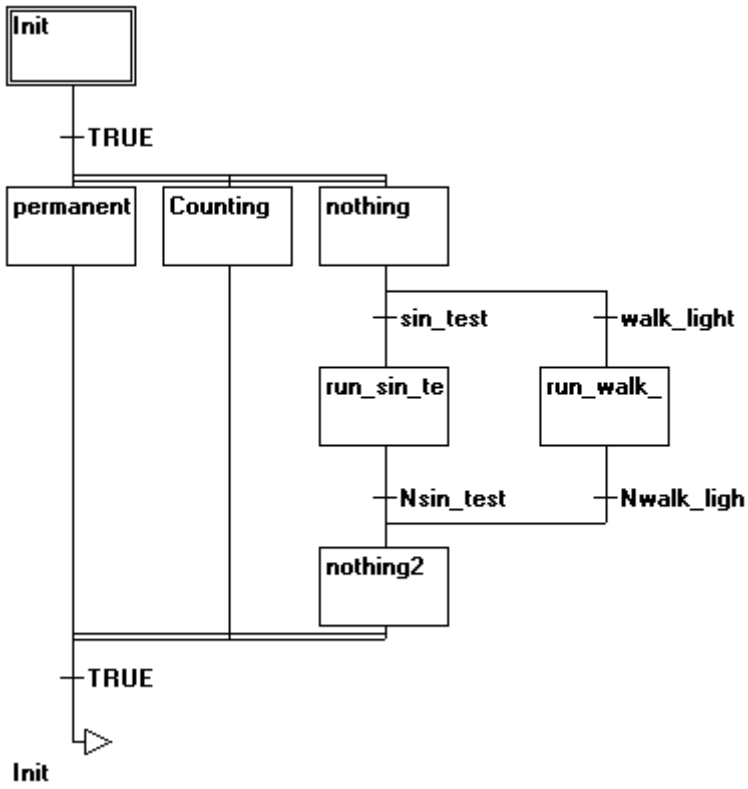


Image 2.8: Network in SFC

Step

A POU written in a Sequential Function Chart consists of a series of steps which are connected with each other through directed connections (transitions).

There are two types of steps.

- The simplified type consists of an action and a flag which shows if the step is active. If the action of a step is implemented, then a small triangle appears in upper right corner of the step.
- An IEC step consists of a flag and one or more assigned actions or boolean variables. The associated actions appear to the right of the step. (see below: IEC Step).

Action

An action can contain a series of instructions in IL or in ST, a lot of networks in FBD or in LD, or again in Sequential Function Chart (SFC).

With the simplified steps an action is always connected to a step. In order to edit an action, click twice with the mouse on the step to which the action belongs. Or select the step and select the menu command **'Extras' 'Zoom Action/Transition'**. In addition, one input or output action per step is possible.

Actions of IEC steps hang in the Object Organizer directly under their SFC-POU and are loaded with a doubleclick or by pressing <Enter> in their editor. New actions can be created with **'Project' 'Add Action'**. Not more than nine actions can be assigned to one IEC step.

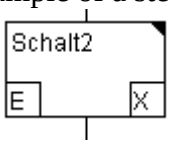
Entry or exit action

Additional to a step action you can add an entry action and an exit action to a step. An entry action is executed only once, right after the step has become active. An exit action is executed only once before the step is deactivated.

A step with entry action is indicated by an "E" in the lower left corner, the exit action by an "X" in the lower right corner.

The entry and exit action can be implemented in any language. In order to edit an entry or exit action, doubleclick in the corresponding corner in the step with the mouse.

Example of a step with entry and exit action:



Transition / Transition condition

Between the steps there are so-called transitions.

A transition condition must have the value TRUE or FALSE. Thus it can consist of either a boolean variable, a boolean address or a boolean constant. It can also contain a series of instructions having a boolean result, either in ST syntax (e.g. (i<= 100) AND b) or in any language desired (see 'Extras' 'Zoom Action/Transition'). But a transition may not contain programs, function blocks or assignments!

Note: Besides transitions, inching mode can also be used to skip to the next step; see SFCtip and SFCtipmode.

Active step

After calling the SFC POU, the action (surrounded by a double border) belonging to the initial stepis executed first. A step, whose action is being executed, is called active. In Online mode active steps are shown in blue.

In a control cycle all actions are executed which belong to active steps. Thereafter the respective following steps of the active steps become active if the transition conditions of the following steps are TRUE. The currently active steps will be executed in the next cycle.

Note: If the active step contains an output action, this will only be executed during the next cycle, provided that the transition following is TRUE.

IEC step

Along with the simplified steps the standard IEC steps in SFC are available.

In order to be able to use IEC steps, you must link the special SFC library **lecsfc.lib** into your project.

Not more than nine actions can be assigned to an IEC step. IEC actions are not fixed as input, step or output actions to a certain step as in the simplified steps, but are stored separately from the steps and can be reused many times within a POU. For

this they must be associated to the single steps with the command '**Extras**''**Associate action**'.

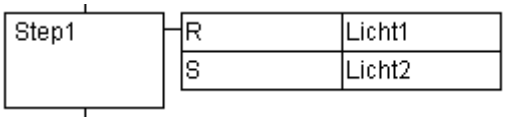
Along with actions, Boolean variables can be assigned to steps.

The activation and deactivation of actions and boolean variables can be controlled using so-called qualifiers. Time delays are possible. Since an action can still be active, if the next step has been processed, for example through the qualifier S (Set), one can achieve concurrent processes.

An associated boolean variable is set or reset with each call of the SFC block. That means, that with each call the value changes from TRUE or FALSE or back again.

The actions associated with an IEC step are shown at the right of the step in a two-part box. The left field contains the qualifier, possibly with time constant, and the right field contains the action name respectively boolean variable name.

An example for an IEC step with two actions:



In order to make it easier to follow the processes, all active actions in online mode are shown in blue like the active steps. After each cycle a check is made to see which actions are active.

Pay attention here also to the restrictions on the use of time-qualifiers in actions that are repeatedly re-used within the same cycle (see “Qualifier”) !

Note: If an action has been inactivated, it will be executed **once more**. That means, that each action is executed at least twice (also an action with qualifier P).

In case of a call first the deactivated actions, then the active actions are executed, in alphabetical order each time.

Whether a newly inserted step is an IEC step depends upon whether the menu command **'Extras' 'Use IEC-Steps'** has been chosen.

In the Object Organizer the actions hang directly underneath their respective SFC POUs. New actions can be created with '**Project**' '**Add Action**'.

In order to use IEC steps you must include in your project the special SFC library Iecsfc.lib .

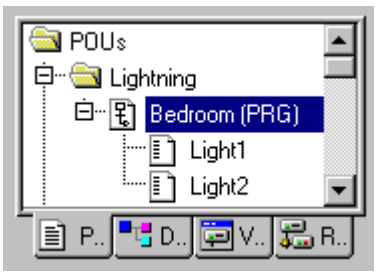


Image 2.9: SFC POU with actions in the Object Organizer

Qualifier

In order to associate the actions with IEC steps the following qualifiers are available:

N	Non-stored	The action is active as long as the step
R	overriding Reset	The action is deactivated
S	Set (Stored)	The action is activated and remains active until a Reset
L	time Limited	The action is activated for a certain time, maximum as long as the step is active
D	time Delayed	The action becomes active after a certain time if the step is still active and then it remains active as long as the step is active.
P	Pulse	The action is executed just one time if the step is active
SD	Stored and time Delayed	The action is activated after a certain time and remains active until a Reset
DS	Delayed and Stored	The action is activated after a certain time as long as the step is still active and remains active up to a Reset
SL	Stored and time limited	The action is activated for a certain time

The qualifiers L, D, SD, DS and SL need a time value in the TIME constant format.

Hinweis: After an action has got deactivated it will be executed **once more**. Thus each action will be executed at least twice (even an action with qualifier P):

Implicit variables in SFC

There are implicitly declared variables in the SFC which can be used.

A flag belongs to each step which stores the state of the step. The step flag (active or inactive state of the step) is called **<StepName>.x** for IEC steps or **<StepName>** for the simplified steps. This Boolean variable has the value TRUE when the associated step is active and FALSE when it is inactive. It can be used in every action and transition of the SFC block.

One can make an enquiry with the variable **<ActionName>.x**. as to whether an IEC action is active or not.

For IEC steps the implicit variables **<StepName>.t** can be used to enquire about the active time of the steps.

Implicit variables can also be accessed by other programs. Example: boolvar1:=sfc1.step1.x; Here, step1.x is the implicit boolean variable representing the state of IEC step step1 in POU sfc1.

SFC Flags

Flags can be used to control the operation of SFC POUs. These flags are created implicitely. You have to define appropriate global or local input resp. output variables to get access to the flags. Example: If in a SFC POU a step is active for a

longer time than defined in the step attributes, then a flag will be set, which is accessible by using a variable "SFCError" (SFCError gets TRUE in this case).

The following flag variables can be defined:

SFCEnableLimit: This variable is of the type BOOL. When it has the value TRUE, the timeouts of the steps will be registered in SFCError. Other timeouts will be ignored.

SFCInit: When this boolean variable has the value TRUE the sequential function chart is set back to the Init step. The other SFC flags are reset too (initialization). The Init step remains active, but is not executed, for as long as the variable has the value TRUE. It is only when SFCInit is again set to FALSE that the block can be processed normally.

SFCReset: This variable, of type BOOL, behaves similarly to SFCInit. Unlike the latter, however, further processing takes place after the initialization of the Init step. Thus for example the SFCReset flag could be re-set to FALSE in the Init step.

SFCQuitError: Execution of the SFC diagram is stopped for as long as this boolean variable has the value TRUE whereby a possible timeout in the variable SFCError is reset. All previous times in the active steps are reset when the variable again assumes the value FALSE. It is a pre-condition that the flag SFCError has been defined also, which registers any timeout in the SFC.

SFCPause: Execution of the SFC diagram is stopped for as long as this boolean variable has the value TRUE.

SFCError: This Boolean variable is TRUE when a timeout has occurred in a SFC diagram. If another timeout occurs in a program after the first one, it will not be registered unless the variable SFCError is reset first. It is a pre-condition that SFCError is defined, if you want to use the other time-controlling flags (SFCErrorStep, SFCErrorPOU, SFCQuitError, SFCErrorAnalyzation).

SFCTrans: This boolean variable takes on the value TRUE when a transition is actuated.

SFCErrorStep: This variable is of the type STRING. If SFCError registers a timeout, in this variable is stored the name of the step which has caused the timeout. It is a pre-condition that the flag SFCError has been defined also, which registers any timeout in the SFC.

SFCErrorPOU: This variable of the type STRING contains the name of the block in which a timeout has occurred. It is a pre-condition that the flag SFCError has been defined also, which registers any timeout in the SFC.

SFCCurrentStep: This variable is of the type STRING. The name of the step is stored in this variable which is active, independently of the time monitoring. In the case of simultaneous sequences the step is stored in the branch on the outer right. No further timeout will be registered if a timeout occurs and the variable SFCError is not reset again.

SFCErrorAnalyzation: This variable, of type STRING, provides the transition expression as well as every variable in an assembled expression which gives a FALSE result for the transition and thus produces a timeout in the preceding step. A requirement for this is declaration of the SFCError flag, which registers the timeout. SFCErrorAnalyzation refers back to a function called AppedErrorString in the

analyzation.lib library. The output string separates multiple components with the symbol “|”.

SFCTip, **SFCTipMode**: This variables of type BOOL allow inching mode of the SFC. When this is switched on by SFCTipMode=TRUE, it is only possible to skip to the next step if SFCTip is set to TRUE. As long as SFCTipMode is set to FALSE, it is possible to skip even over transitions.

Alternative branch

Two or more branches in SFC can be defined as alternative branches. Each alternative branch must begin and end with a transition. Alternative branches can contain parallel branches and other alternative branches. An alternative branch begins at a horizontal line (alternative beginning) and ends at a horizontal line (alternative end) or with a jump.

If the step which precedes the alternative beginning line is active, then the first transition of each alternative branch is evaluated from left to right. The first transition from the left whose transition condition has the value TRUE is opened and the following steps are activated (see 'Active step').

Parallel branch

Two or more branches in SFC can be defined as parallel branches. Each parallel branch must begin and end with a step. Parallel branches can contain alternative branches or other parallel branches. A parallel branch begins with a double line (parallel beginning) and ends with a double line (parallel end) or with a jump. It can be provided with a jump label.

If the parallel beginning line of the previous step is active and the transition condition after this step has the value TRUE, then the first steps of all parallel branches become active (see 'Active step'). These branches are now processed parallel to one another. The step after the parallel end line becomes active when all previous steps are active and the transition condition before this step produces the value TRUE.

Jump

A jump is a connection to the step whose name is indicated under the jump symbol. Jumps are required because it is not allowed to create connections which lead upward or cross each other.

2.2.4 Function Block Diagram (FBD)

The Function Block Diagram is a graphically oriented programming language. It works with a list of networks whereby each network contains a structure which represents either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction.

An example of a typical network in the Function Block Diagram as it could appear in CoDeSys :

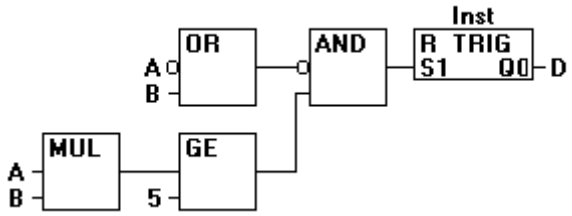


Image 2.10: Network in Function Block Diagram

See also Chapter 5.3.1 for more information and see also the Continuous Function Chart Editor.

2.2.5 The Continuous Function Chart Editor (CFC)

The continuous function chart editor does not operate like the function block diagram FBD with networks, but rather with freely placeable elements. This allows feedback, for example.

An example of a network in the continuous function chart editor, as it would typically appear in **CoDeSys**:

Image 2.11: A network in the continuous function chart editor

See also Chapter 5.3.4 for more information.

2.2.6 Ladder Diagram (LD)

The Ladder Diagram is also a graphics oriented programming language which approaches the structure of an electric circuit.

On the one hand, the Ladder Diagram is suitable for constructing logical switches, on the other hand one can also create networks as in FBD. Therefore the LD is very useful for controlling the call of other POUs.

The Ladder Diagram consists of a series of networks. A network is limited on the left and right sides by a left and right vertical current line. In the middle is a circuit diagram made up of contacts, coils, and connecting lines.

Each network consists on the left side of a series of contacts which pass on from left to right the condition "ON" or "OFF" which correspond to the Boolean values TRUE and FALSE. To each contact belongs a Boolean variable. If this variable is TRUE, then the condition is passed from left to right along the connecting line. Otherwise the right connection receives the value OFF.

Example of a typical network in the Ladder Diagram as it could appear in CoDeSys :

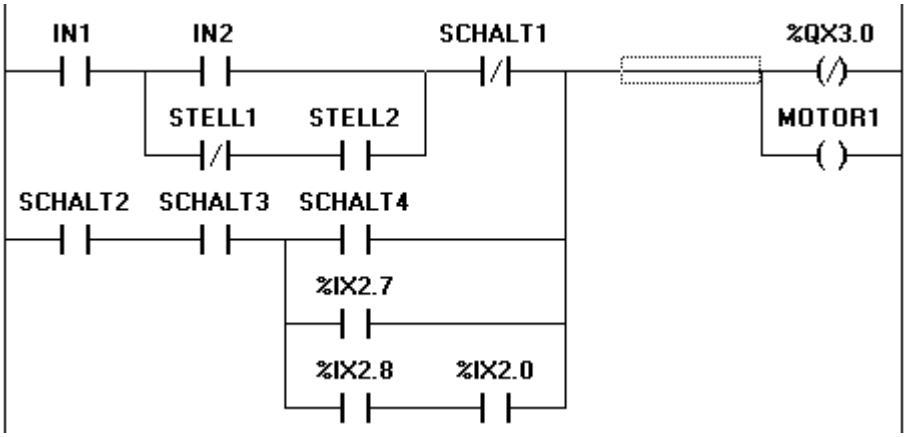


Image 2.12: Network in a Ladder Diagram made up of Contacts and Coils

Contact

Each network in LD consists on the left side of a network of *contacts* (contacts are represented by two parallel lines: | |) which from left to right show the condition "On" or "Off".

These conditions correspond to the Boolean values TRUE and FALSE. A Boolean variable belongs to each contact. If this variable is TRUE, then the condition is passed on by the connecting line from left to right, otherwise the right connection receives the value "Out".

Contacts can be connected in parallel, then one of the parallel branches must transmit the value "On" so that the parallel branch transmits the value "On"; or the contacts are connected in series, then contacts must transmit the condition "On" so that the last contact transmits the "On" condition. This therefore corresponds to an electric parallel or series circuit.

A contact can also be negated, recognizable by the slash in the contact symbol: |/|. Then the value of the line is transmitted if the variable is FALSE.

Coil

On the right side of a network in LD there can be any number of so-called coils which are represented by parentheses:**( )**. They can only be in parallel. A coil transmits the value of the connections from left to right and copies it in an appropriate Boolean variable. At the entry line the value ON (corresponds to the Boolean variable TRUE) or the value OFF (corresponding to FALSE) can be present.

Contacts and coils can also be negated (in the example the contact SWITCH1 and the coil %QX3.0 is negated). If a coil is negated (recognizable by the slash in the coil symbol: **(/)**), then it copies the negated value in the appropriate Boolean variable. If a contact is negated, then it connects through only if the appropriate Boolean value is FALSE.

Function blocks in the Ladder Diagram

Along with contacts and coils you can also enter function blocks and programs In the network they must have an input and an output with Boolean values and can be used at the same places as contacts, that is on the left side of the LD network

Set/Reset coils

Coils can also be defined as set or reset coils. One can recognize a set coil by the "S" in the coil symbol: **(S)**) It never writes over the value TRUE in the appropriate Boolean variable. That is, if the variable was once set at TRUE, then it remains so.

One can recognize a reset coil by the "R" in the coil symbol: **(R)**) It never writes over the value FALSE in the appropriate Boolean variable: If the variable has been once set on FALSE, then it remains so.

LD as FBD

When working with LD it is very possible that you will want to use the result of the contact switch for controlling other POUs. On the one hand you can use the coils to put the result in a global variable which can then be used in another place. You can, however, also insert the possible call directly into your LD network. For this you introduce a POU with EN input.

Such POUs are completely normal operands, functions, programs, or function blocks which have an additional input which is labeled with EN. The EN input is always of the BOOL type and its meaning is: The POU with EN input is evaluated when EN has the value TRUE.

An EN POU is wired parallel to the coils, whereby the EN input is connected to the connecting line between the contacts and the coils. If the ON information is transmitted through this line, this POU will be evaluated completely normally.

Starting from such an EN POU, you can create networks similar to FBD.

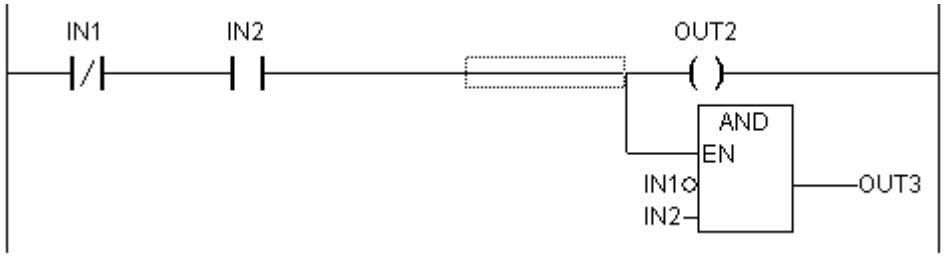


Image 2.13: Part of a LD Network with an EN POU

2.3 Debugging, Online Functions

Sampling Trace

The Sampling Trace allows you to trace the value sequence of variables, depending upon the so-called trigger event. This is the rising edge or falling edge of a

previously defined Boolean variable (trigger variable). **CoDeSys** permits the tracing of up to 20 variables. 500 values can be traced for each variable.

Debugging

The debugging functions of CoDeSys make it easier for you to find errors.

In order to debug, run the command **'Project' 'Options'** and in the dialog box that pops up under **Build** options select activate option **Debugging**.

Breakpoint

A breakpoint is a place in the program at which the processing is stopped. Thus it is possible to look at the values of variables at specific places within the program.

Breakpoints can be set in all editors. In the text editors breakpoints are set at line numbers, in FBD and LD at network numbers, in CFC at POUs and in SFC at steps. No breakpoints can be set in function block instances.

Single step

Single step means:

- In IL: Execute the program until the next CAL, LD or JMP command.
- In ST: Execute the next instruction.
- In FBD, LD: Execute the next network.
- In SFC: Continue the action until the next step.

By proceeding step by step you can check the logical correctness of your program.

Single Cycle

If Single cycle has been chosen, then the execution is stopped after each cycle.

Change values online

During operations variables can be set once at a certain value (write value) or also described again with a certain value after each cycle (force value). In online mode one also can change the variable value by double click on the value. By that boolean variables change from TRUE to FALSE or the other way round, for each other types of variables one gets the dialog **Write Variable xy**, where the actual value of the variable can be edited.

Monitoring

In Online mode, all displayable variables are read from the controller and displayed in real time. You will find this display in the declarations and program editor; you can also read out current values of variables in the watch and receipt manager and in a visualization. If variables from instances of function blocks are to be monitored, the corresponding instance must first be opened (see in this connection Chapter 4.4: Create Objects).

In monitoring VAR_IN_OUT variables, the de-referenced value is output.

In monitoring pointers, both the pointer and the de-referenced value are output in the declaration portion. In the program portion, only the pointer is output:

+ --pointervar = ‘<‘pointervalue‘>‘

POINTERs in the de-referenced value are also displayed accordingly. With a simple click on the cross or a double-click on the line, the display is either expanded or truncated.

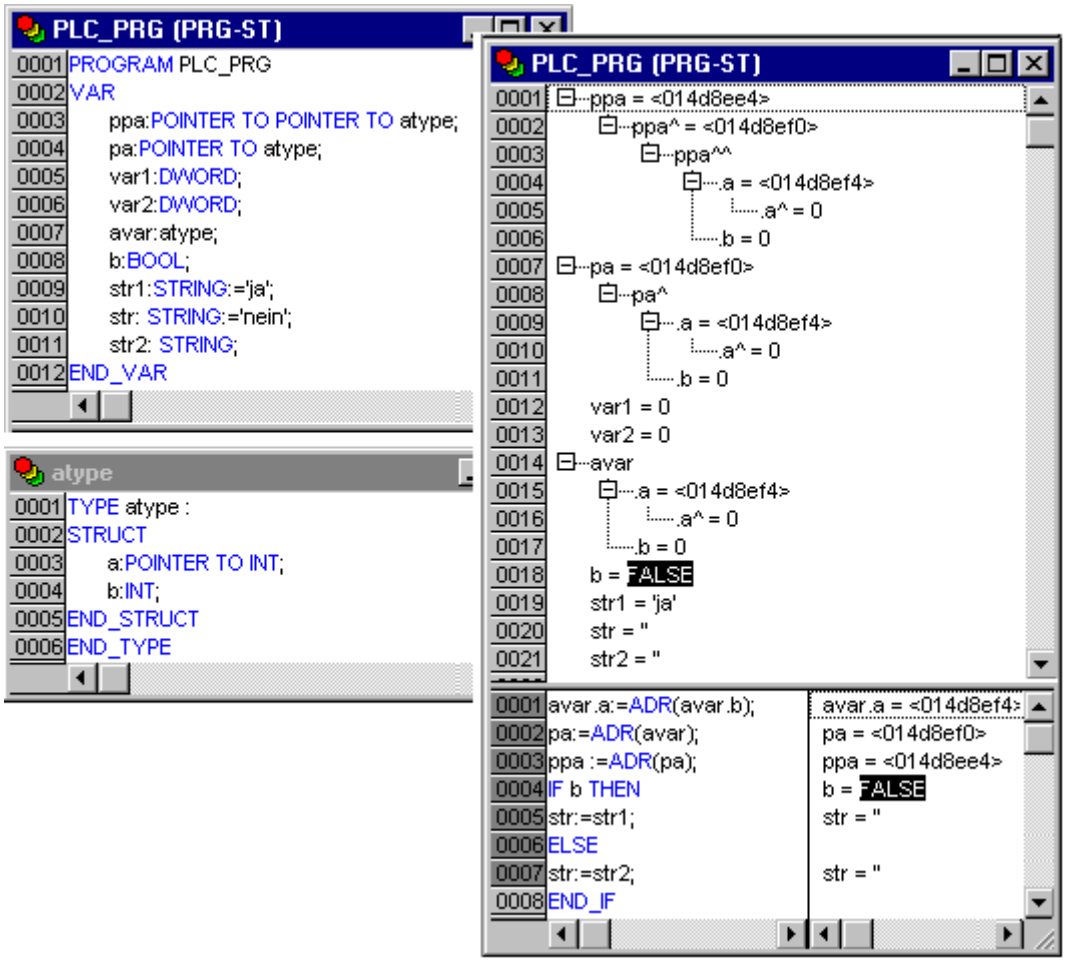


Image 2.14: Example for Monitoring of Pointers

In the implementations, the value of the pointer is displayed. For de-referencing, however, the de-referenced value is displayed.

Monitoring of ARRAY components: In addition to array components indexed by a constant, components are also displayed which are indexed by a variable:

anarray[1] = 5
anarray[i] = 1

If the index consists of an expression (e.g. [i+j] or [i+1]), the component can not be displayed.

Simulation

During the simulation the created PLC program is not processed in the PLC, but rather in the calculator on which **CoDeSys** is running. All online functions are available. That allows you to test the logical correctness of your program without PLC hardware.

Log

The log chronologically records user actions, internal processes, state changes and exceptions during Online mode processing. It is used for monitoring and for error tracing (see Chapter 4.6, Online Functions).

2.4 The Standard

The standard IEC 61131-3 is an international standard for programming languages of Programmable Logic Controllers.

The programming languages offered in **CoDeSys** conform to the requirements of the standard.

According to this standard, a program consists of the following elements:

- Structures (see Appendix Appendix B: , Data Types)
- POUs
- Global Variables

The processing of a CoDeSys program starts with the special POU PLC_PRG. The POU PLC_PRG can call other POUs.

3 We Write a Little Program

3.1 Controlling a Traffic Signal Unit

Let us now start to write a small example program. It is for a simple traffic signal unit which is supposed to control two traffic signals at an intersection. The red/green phases of both traffic signals alternate and, in order to avoid accidents, we will insert yellow or yellow/red transitional phases. The latter will be longer than the former.

In this example you will see how time dependent programs can be shown with the language resources of the IEC1131-3 standard, how one can edit the different languages of the standard with the help of **CoDeSys**, and how one can easily connect them while becoming familiar with the simulation of **CoDeSys**.

Create POU

Starting always is easy: Start CoDeSys and choose **'File' 'New'**.

In the dialog box which appears, the first POU has already been given the default name PLC_PRG. Keep this name, and the type of POU should definitely be a program. Each project needs a program with this name. In this case we choose as the language of this POU the Continuous Function Chart Editor (CFC)

Now create three more objects with the command **'Project' 'Object Add'** with the menu bar or with the context menu (press right mouse button in the Object Organizer). A program in the language Sequential Function Chart (SFC named SEQUENCE, a function block in the language Function Block Diagram (FBD) named TRAFFICSIGNAL, along with a POU WAIT, also of the type function block, which we want to program as an Instruction List (IL.

What does TRAFFICSIGNAL do?

In the POU TRAFFICSIGNAL we will assign the individual trafficsignal phases to the lights, i.e. we will make sure that the red light is lit red in the red phase and in the yellow/red phase, the yellow light in the yellow and yellow/red phases, etc.

What does WAIT do?

In WAIT we will program a simple timer which as input will receive the length of the phase in milliseconds, and as output will produce TRUE as soon as the time period is finished.

What does SEQUENCE do?

In SEQUENCE all is combined so that the right light lights up at the right time for the desired time period.

What does PLC_PRG do?

In PLC_PRG the input start signal is connected to the traffic lights' sequence and the "color instructions" for each lamp are provided as outputs.

"TRAFFICSIGNAL" declaration

Let us now turn to the POU TRAFFICSIGNAL. In the declaration editor you declare as input variable (between the keywords VAR_INPUT and END_VAR) a variable named STATUS of the type INT. STATUS will have four possible conditions, that is one for the TRAFFICSIGNAL phases green, yellow, yellow/red andred.

Correspondingly our TRAFFICSIGNAL has three outputs, that is RED, YELLOW and GREEN. You should declare these three variables. Then the declaration part of our function block TRAFFICSIGNAL will look like this:

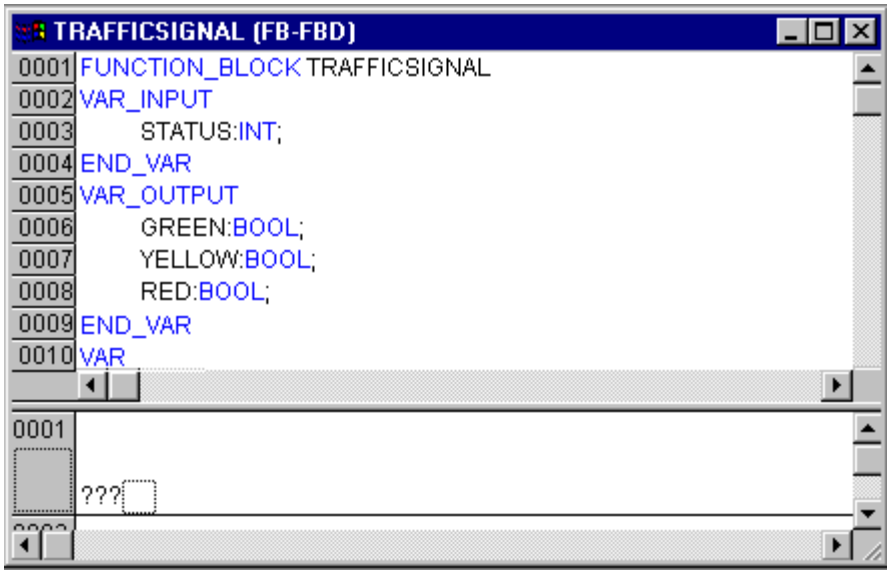
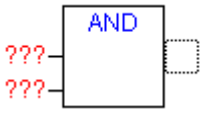


Image 3.1: Function block TRAFFICSIGNAL, declaration part

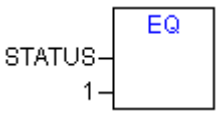
"TRAFFICSIGNAL" body

Now we determine the values of the output variables depending on the input STATUS of the POU. To do this go into the body of the POU. Click on the field to the left beside the first network (the gray field with the number 1). You have now selected the first network. Choose the menu item **'Insert' 'Box'**.

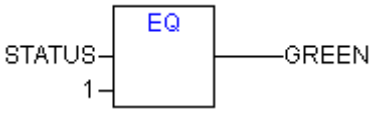
In the first network a box is inserted with the operator AND and two inputs:



Click on the text AND so that it appears selected and change the text into EQ. Select then for each of the two inputs the three question marks and overwrite them with "STATUS" respectively "1":



Click now on a place behind the EQ Box. Now the output of the EQ operation is selected. Choose **'Insert' 'Assign'.** Change the three question marks ??? to GREEN. You now have created a network with the following structure:



STATUS is compared with 1, the result is assigned to GREEN. This network thus switches to GREEN if the preset state value is 1.

For the other TRAFFICSIGNAL colors we need two more networks. To create the first one execute command **'Insert' 'Network (after)'** and insert an EQ-Box like described above. Then select the output pin of this box and use again command **'Insert' 'Box'**. In the new box replace "AND" by "OR". Now select the first output pin of the OR-box and use command **'Insert' 'Assign'** to assign it to "GELB". Select the second input of the OR-box by a mouse-click on the horizontal line next to the three question marks, so that it appears marked by a dotted rectangle. Now use **'Insert' 'Box'** to add a further EQ-box like described above. Finally the network should look like shown in the following:

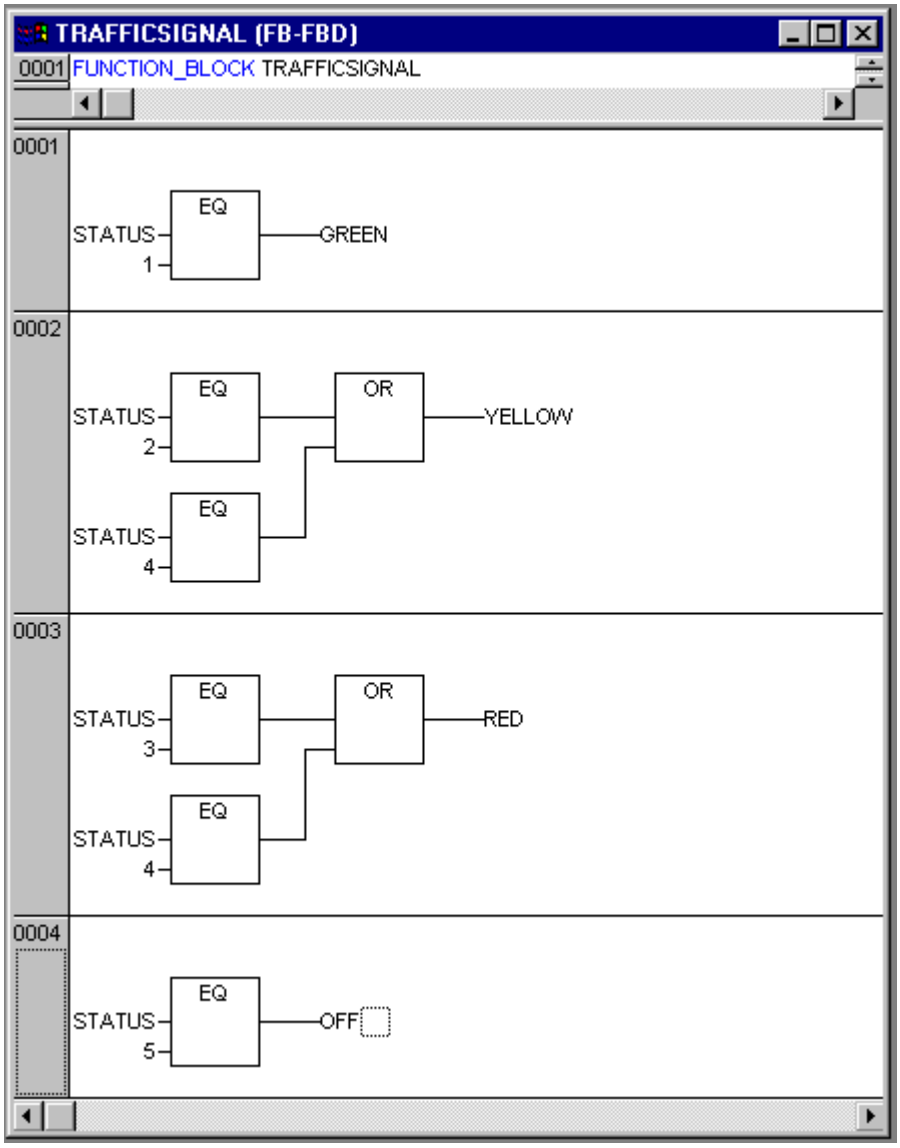


Image 3.2: Function block TRAFFICSIGNAL, instruction part

In order to insert an operator in front of another operator, you must select the place where the input to which you want to attach the operator feeds into the box.

Then use the command **'Insert' 'Box'**. Otherwise you can set up these networks in the same way as the first network.

Now our first POU has been finished. TRAFFICSIGNAL, according to the input of the value STATUS, controls whichever light color we wish.

Connecting the standard.lib

For the timer in the POU WAIT we need a POU from the standard library. Therefore, open the library manager with **'Window' 'Library Manager**. Choose **'Insert' 'Additional library'**. The dialog box appears for opening files. From the list of the libraries choose standard.lib.

"WAIT" declaration

Now let us turn to the POU WAIT. This POU is supposed to become a timer with which we can determine the length of the time period of each TRAFFICSIGNAL phase. Our POU receives as input variable a variable TIME of the type TIME, and as output it produces a Boolean value which we want to call OK and which should be TRUE when the desired time period is finished. We set this value with FALSE by inserting at the end of the declaration (before the semicolon, however) " := FALSE ".

For our purposes we need the POU TP, a clock generator. This has two inputs (IN, PT) and two outputs (Q, ET). TP does the following:

As long as IN is FALSE, ET is 0 and Q is FALSE. As soon as IN provides the value TRUE, the time is calculated at the output ET in milliseconds. When ET reaches the value PT, then ET is no longer counted. Meanwhile Q produces TRUE as long as ET is smaller than PT. As soon as the value PT has been reached, then Q produces FALSE again. In addition you will find a short description of all POUs from the standard library in the appendix.

In order to use the POU TP in the POU WAIT we must create a local instance from TP. For this we declare a local variable ZAB (for elapsed time) of the type TP (between the keywords VAR, END_VAR).

The declaration part of WAIT thus looks like this:

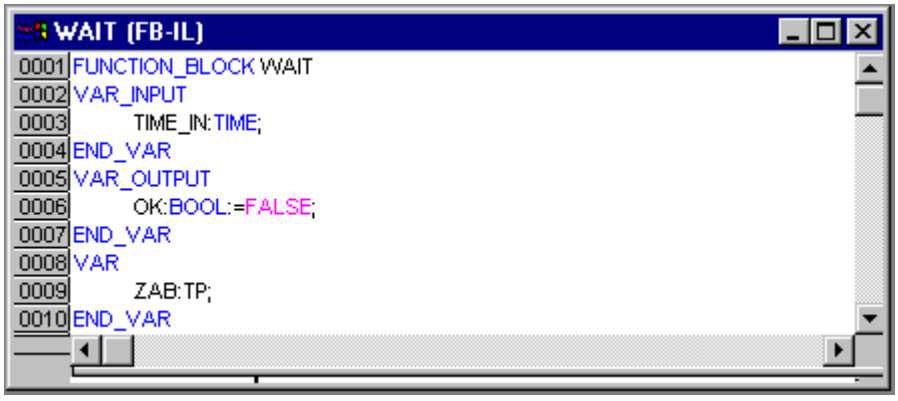


Image 3.3: Function Block WAIT, Declaration Part

"WAIT" body

In order to create the desired timer, the body of the POU must be programmed as follows:

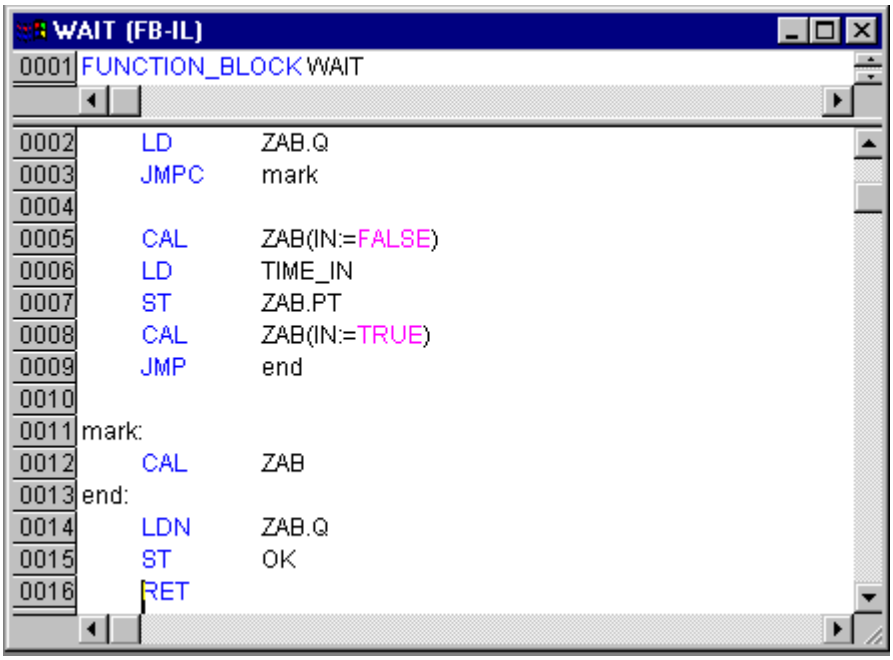


Image 3.4: Function Block WAIT, Instruction Part

At first it is checked whether Q has already been set at TRUE (as though the counting had already been executed), in this case we change nothing with the occupation of ZAB, but we call the function block ZAB without input (in order to check whether the time period is already over).

Otherwise we set the variable IN in ZAB at FALSE, and therefore at the same time ET at 0 and Q at FALSE. In this way all variables are set at the desired initial condition. Now we assign the necessary time from the variable TIME into the variable PT, and call ZAB with IN:=TRUE. In the function block ZAB the variable ET is now calculated until it reaches the value TIME, then Q is set at FALSE.

The negated value of Q is saved in OK after each execution of WAIT. As soon as Q is FALSE, then OK produces TRUE.

The timer is finished at this point. Now it is time to combine our two function blocks WAIT and TRAFFICSIGNAL in the main program PLC_PRG.

"SEQUENCE" first expansion level

First we declare the variables we need. They are: an input variable START of the type BOOL, two output variables TRAFFICSIGNAL1 and TRAFFICSIGNAL2 of the type INT and one of the type WAIT (DELAY as delay). The program SEQUENCE now looks like shown here:

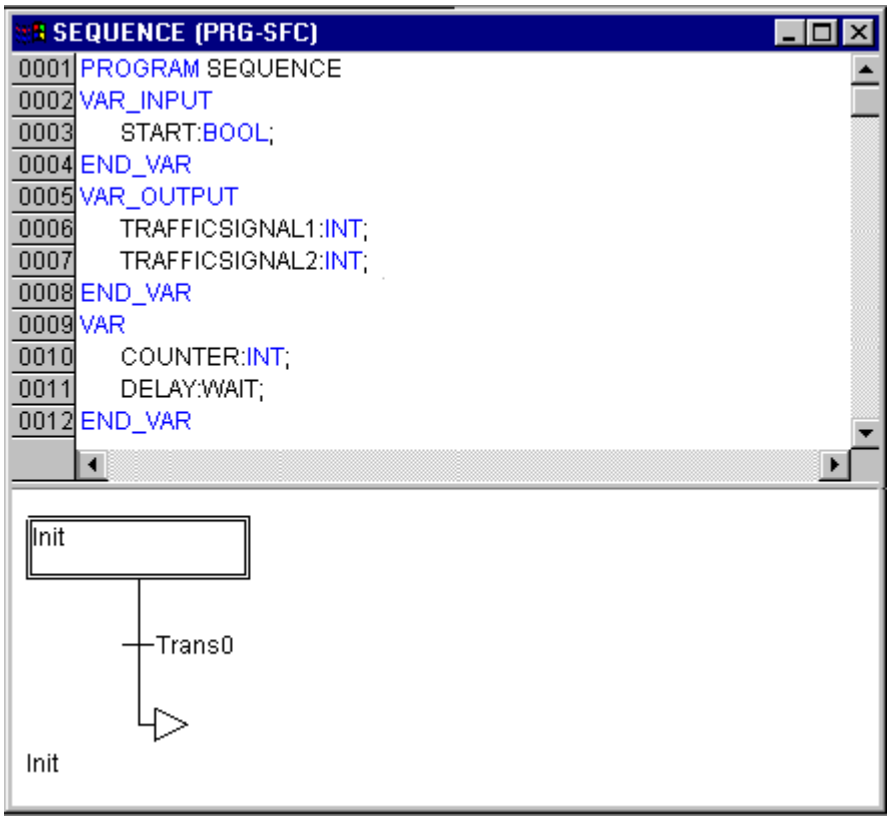


Image 3.5: Program Sequence, First Expansion Level, Declaration Part

Create a SFC diagram

The beginning diagram of a POU in SFC always consists of an action "Init" of a following transition "Trans0" and a jump back to Init. We have to expand that.

Before we program the individual action and transitions let us first determine the structure of the diagrams. We need one step for each TRAFFICSIGNAL phase. Insert it by marking Trans0 and choosing **'Insert' 'Step transition (after)'**. Repeat this procedure three more times.

If you click directly on the name of a transition or a step, then this is marked and you can change it. Name the first transition after Init "START", and all other transitions "DELAY.OK".

The first transition switches through when START is TRUE and all others switch through when DELAY in OK produces TRUE, i.e. when the set time period is finished.

The steps (from top to bottom) receive the names Switch1, Green2, Switch2, Green1, whereby Init of course keeps its Name. "Switch" should include a yellow phase, at Green1 TRAFFICSIGNAL1 will be green, at Green2 TRAFFICSIGNAL2 will be green. Finally change the return address of Init after Switch1. If you have done everything right, then the diagram should look like in the following image:

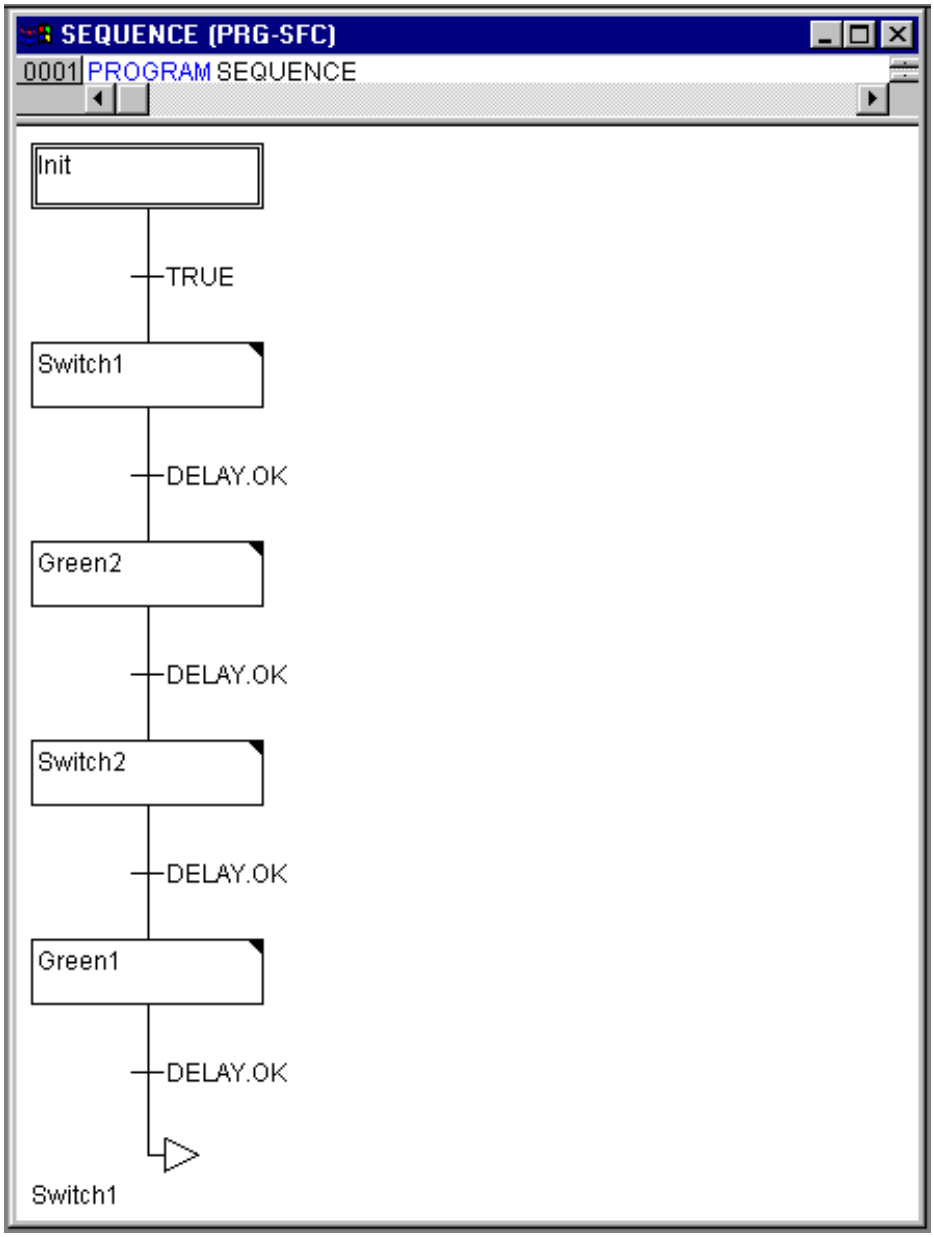


Image 3.6: Program SEQUENCE, First Expansion Level, Instruction Part

Now we have to finish the programming of the individual steps. If you doubleclick on the field of a step, then you get a dialog for opening a new action. In our case we will use IL (Instruction List).

Actions and transition conditions

In the action of the step **Init** the variables are initialized, the STATUS of TRAFFICSIGNAL1 should be 1 (green). The state of TRAFFICSIGNAL2 should be 3 (red). The action Init then looks like in the following image:

```
Action Init (IL)
0001   LD    1
0002   ST    TRAFFICSIGNAL1
0003   LD    3
0004   ST    TRAFFICSIGNAL2
```

Image 3.7: Action Init

Switch1 changes the sate of TRAFFICSIGNAL1 to 2 (yellow), and that of TRAFFICSIGNAL2 to 4 (yellow-red). In addition, a time delay of 2000 milliseconds is set. The action is now as follows:

```
Action Switch1
0001   LD    2
0002   ST    TRAFFICSIGNAL1
0003   LD    4
0004   ST    TRAFFICSIGNAL 2
0005   CAL   DELAY(TIME_IN:=t#2s)
```

Image 3.8: Action Switch1

With **Green2** TRAFFICSIGNAL1 is red (STATUS:=3), TRAFFICSIGNAL2 green (STATUS:=1), and the delay time is 5000 milliseconds.

```
Action Green2
0001   LD    3
0002   ST    TRAFFICSIGNAL1
0003   LD    1
0004   ST    TRAFFICSIGNAL2
0005   CAL   DELAY(TIME_IN:=t#5s)
```

Image 3.9: Action Green2

At **Switch2** the STATUS of TRAFFICSIGNAL1 changes to 4 (yellow-red), that of TRAFFICSIGNAL2 to 2 (yellow). A time delay of 2000 milliseconds is now set.

```
Action Switch2 (IL)
0001   LD    4
0002   ST    TRAFFICSIGNAL1
0003   LD    2
0004   ST    TRAFFICSIGNAL2
0005   CAL   DELAY(TIME_IN:=t#2s)
```

Image 3.10: Action Switch2

With **Green1** TRAFFICSIGNAL1 is green (STATUS:=1), TRAFFICSIGNAL2 is red (STATUS:=3), and the time delay is set to5000 milliseconds.

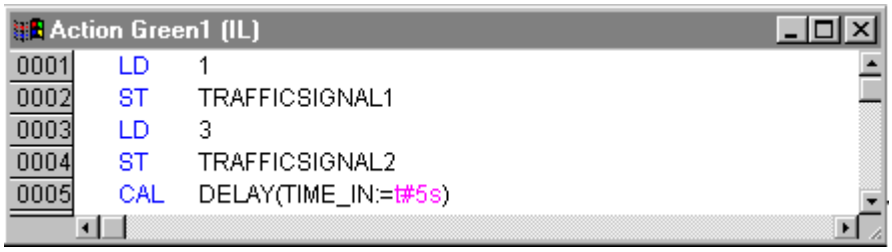


Image 3.11: Action Green1

The first expansion phase of our program is completed.. Now you can test the POU ABLAUF in simulation mode. Compile the project: **'Project' 'Build'**. In the message window you should get "0 Errors, 0 Warnings". Now check if option 'Online' 'Simulation' is activated and use command **'Online' 'Login'** to get into simulation mode. Start the program with **'Online' 'Start'**. Open POU ABLAUF by a double-click on "ABLAUF" in the Object Organizer. The program is started now, but to get it run, variable START must be TRUE. Later this will be set by PLC_PRG but at the moment we have to set it manually within the POU. To do that, perform a double-click on the line in the declaration part, where START is defined (START=FALSE). This will set the option "<:=TRUE>" behind the variable in turquoise color. Now select command 'Online' 'Write values' to set this value. Thereupon START will be displayed in blue color in the sequence diagram and the processing of the steps will be indicated by a blue mark of the currently active step.

When you have finished this intermediate test use command **'Online' 'Logout'** to leave the simulation mode and to continue programming.

"SEQUENCE" second expansion level

In order to ensure that our diagram has at least one alternative branch, and so that we can turn off our traffic light unit at night, we now include in our program a counter which, after a certain number of TRAFFICSIGNAL cycles, turns the unit off.

At first we need a new variable COUNTER of the type INT. Declare this as usual in the declaration part of PLC_PRG, and initialize it in Init with 0.

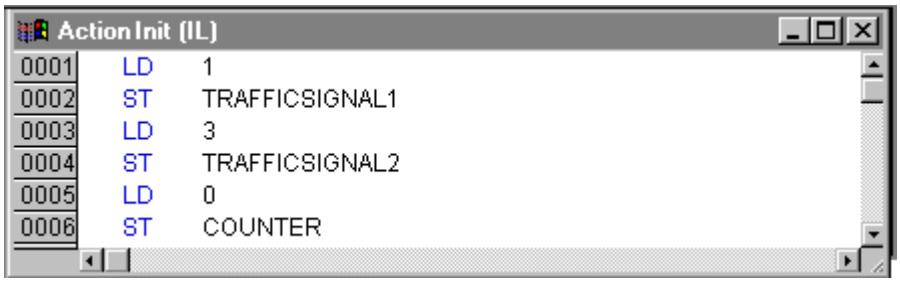


Image 3.12: Action Init, Second Version

Now select the transition after Switch1 and insert a step and then a transition. Select the resulting transition and insert an alternative branch to its left. After the left transition insert a step and a transition. After the resulting new transition insert a jump after Switch1.

Name the new parts as follows: the upper of the two new steps should be called "Count" and the lower "Off". The transitions are called (from top to bottom and from

left to right) EXIT, TRUE and DELAY.OK. The new part should look like the part marked with the black border in the following image:

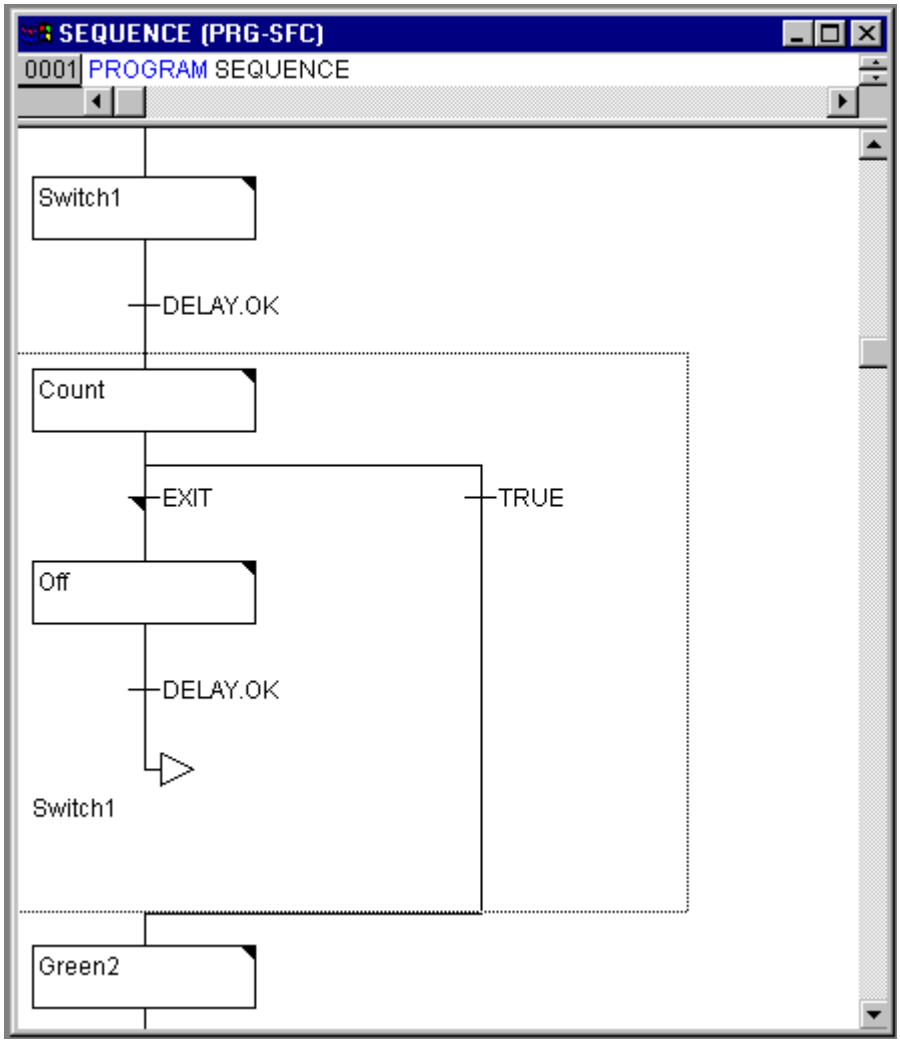


Image 3.13: Program SEQUENCE, Second Expansion Level, Instruction Part

Now two new actions and a new transition condition are to be implemented. At the step Count the variable COUNTER is increased by one:

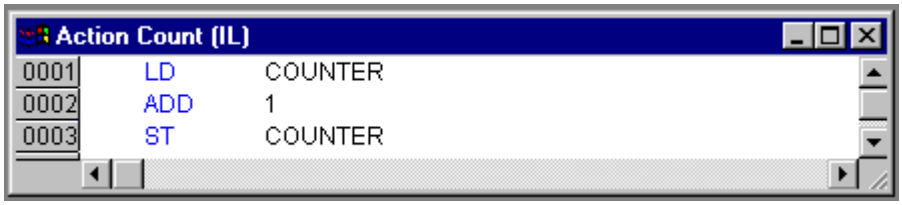


Image 3.14: Action Count

The EXIT transition checks whether the counter is greater than a certain value, for example 7:

Image 3.15: Transition EXIT

At Off the state of both lights is set at 5(OFF), (or each other number not equal 1,2,3 or 4) the COUNTER is reset to 0, and a time delay of 10 seconds is set:

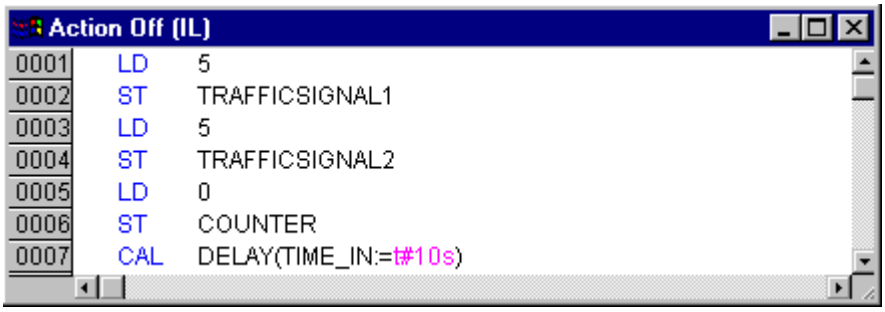


Image 3.16: Action Off

The result

In our hypothetical situation, night falls after seven TRAFFICSIGNAL cycles, for ten seconds the TRAFFICSIGNAL turns itself off, then we have daylight again, the traffic light unit turns itself on again, and the whole process starts again from the beginning. If you like, do another test of the current version of your program in simulation mode before we go on to create the POU PLC_PRG.

PLC_PRG

We have defined and correlated the time sequencing of the phases for both sets of traffic lights in the block SEQUENCE. Since, however, we see the traffic lights system as a module of a bus system, e.g. CAN bus, we have to make input and output variables available in the block PLC_PRG. We want to start-up the traffic lights system over an ON switch and we want to send each of the six lamps (each traffic light red, green, yellow) the corresponding ”signal command” for each step of the SEQUENCE. We are now declaring appropriate Boolean variables for these six outputs and one input, before we create the programme in the editor, and are allocating them, at the same time, to the corresponding IEC addresses.

The next step is declare the variables Light1 and Light2 of the type Phases in the declaration editor.

Image 3.17: Declaration LIGHT1 and LIGHT2

These deliver the Boolean value of each of the six lights to the above mentioned six outputs for each step of the block SEQUENCE. We are not, however, declaring the output variables which are foreseen within the PLC_PRG block but under Resources for Global Variables instead. The Boolean input variable IN, which is used to set the variable START in the block SEQUENCE to TRUE, can be set in the same way. ON is also allocated to an IEC address.

Select the 'Resources' tab and open the list 'Global Variables'.

Make the declaration as follows:

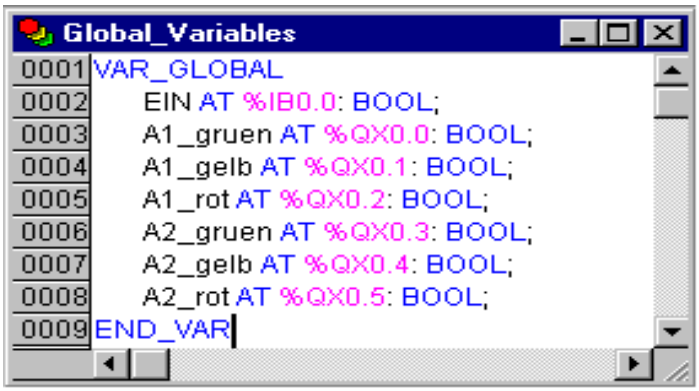

Global_Variables
0001 VAR_GLOBAL
0002 EIN AT %IB0.0: BOOL;
0003 A1_gruen AT %QX0.0: BOOL;
0004 A1_gelb AT %QX0.1: BOOL;
0005 A1_rot AT %QX0.2: BOOL;
0006 A2_gruen AT %QX0.3: BOOL;
0007 A2_gelb AT %QX0.4: BOOL;
0008 A2_rot AT %QX0.5: BOOL;
0009 END_VAR

Image 3.18: Declaration of the Input-/Output Variables for a CAN-Configuration

The name of the variable (e.g. IN) is followed, after AT, by a percent sign which begins the IEC address. I stands for input, Q for output, B (used in this example) stands for byte and the individual bits of the module are addressed using 0.0 (0.1, 0.2, etc.). We will not do the needed controller configuration here in this example, because it depends on which target package you have available on your computer. Please see Chapter 6.6, PLC configuration, for further information.

We now want to finish off the block PLC_PRG.

For this we go into the editor window. We have selected the Continuous Function Chart editor and we consequently obtain, under the menu bar, a CFC symbol bar with all of the available elements (see Chapter 3.3.4, The Continuous Function Chart Editor).

Click on the right mouse key in the editor window and select the element **Box**. Click on the text AND and write ”SEQUENCE” instead. This brings up the block SEQUENCE with all of the already defined input and output variables. Insert two further block elements which you name PHASES. Phases is a function block and this causes you to obtain three red question marks over the block which you replace with the already locally declared variables LIGHT1 and LIGHT2. Now set an element of the type **Input**, which award the title ON and six elements of the type **Output** which you award variable names to, as described, namely L1_green, L1_yellow, L1_red, L2_green, L2_yellow, L2_red.

All of the elements of the programme are now in place and you can connect the inputs and outputs, by clicking on the short line at the input/output of an element and dragging this with a constantly depressed mouse key to the input/output of the desired element.

Your program should finally look like the example shown here.

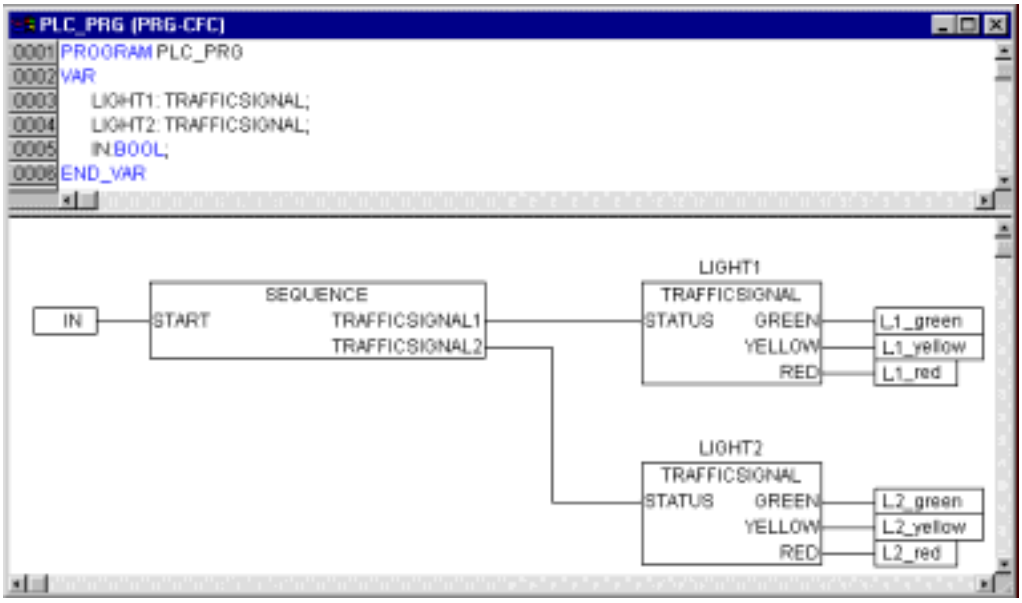


Image 3.19: PLC_PRG, Declaration and presentation with the continuous function chart editor

TRAFFICSIGNAL simulation

Now test your program in simulation mode. Compile (**'Project' 'Build'**) and load (**'Online' 'Login'**) it. Start the program by **'Online' 'Start'**, then set variable ON to TRUE, e.g. by a double-click on the entry "ON" in the input box of the CFC editor. This will mark the variable as prepared to be set to <TRUE>. Then press <Strg><F7> or command 'Online' 'Write values', to set the value. Now variable START in ABLAUF (which we had set to TRUE manually in the first extension level of the program) gets this value by variable ON, which is used in PLC_PRG. This will make run the traffic light cycles. PLC_PRG has changed to a monitoring window. Click twice on the plus sign in the declaration editor, the variable display drops down, and you can see the values of the individual variables.

3.2 Visualizing a Traffic Signal Unit

With the visualization of **CoDeSys** you can quickly and easily bring project variables to life. You find an complete description of the visualization in chapter 8. We will now plot two traffic signals and an ON-Switch for our traffic light unit which will illustrate the switching process.

Creating a new visualization

In order to create a visualization you must first select the range of **Visualization** in the Object Organizer. First click on the lower edge of the window on the left side with the **POU** on the register card with this symbol and the name **Visualization**. If you now choose the command **'Project' 'Object Add'**, then a dialog box opens.

Image 3.20: Dialog Box for Opening a New Visualization

Enter here any name. When you confirm the dialog with **OK**, then a window opens in which you can set up your new visualization.

Insert element in Visualization

For our TRAFFICSIGNAL visualization you should proceed as follows:

- Give the command **'Insert' 'Ellipse'** and try to draw a medium sized circle (∅2cm). For this click in the editor field and draw with pressed left mouse button the circle in its length.
- Now doubleclick the circle. The dialog box for editing visualization elements opens
- Choose the category 'Variables' and in the field 'Change color' enter the variable name ".L1_red" or "L1_red". That means that the global variable L1_red will cause the color change as soon as it is set to TRUE. The dot before the variable name indicates that it is a global variable, but it is not mandatory.

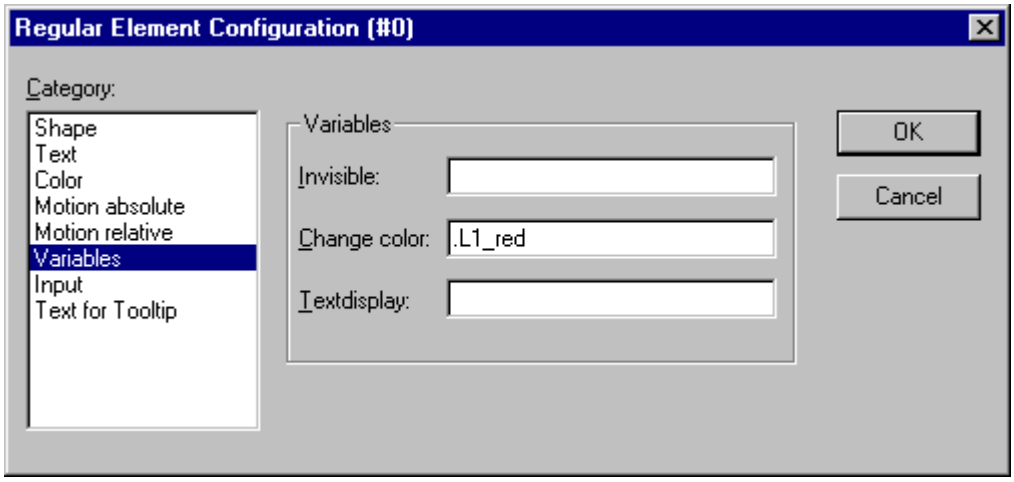


Image 3.21: Visualization Dialog Box Variables

- Then choose the category **Color** and click on the button **Inside** in the area **Color**. Choose as neutral a color as possible, such as black.

- Now click on the button **within** in the area **Alarm color** and choose the red which comes closest to that of a red light.

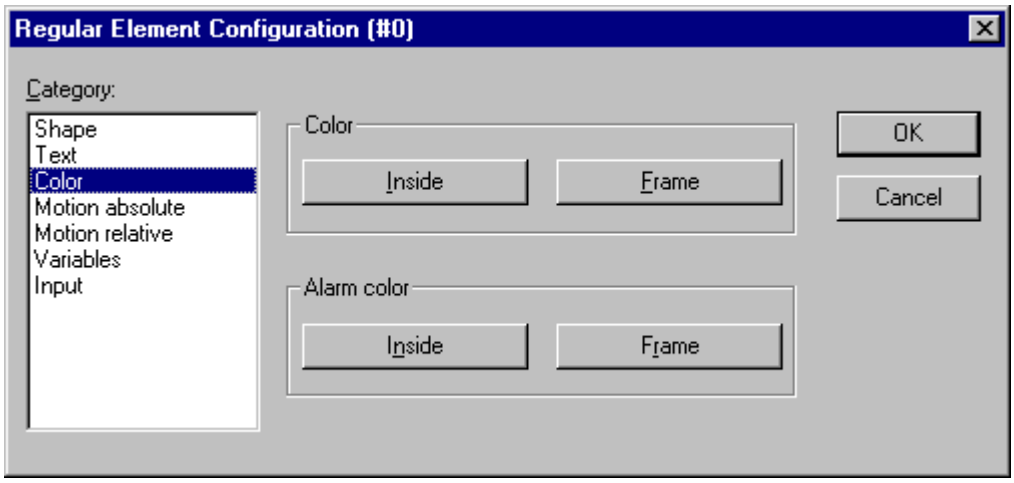


Image 3.22: Visualization Configuration Dialog Box (Color category)

The resulting circle will normally be black, and when the variable RED from TRAFFICSIGNAL1 is TRUE, then its color will change to red. We have therefore created the first light of the first TRAFFICSIGNAL!

The other traffic lights

Now enter the commands 'Edit' 'Copy' (<Ctrl>+<C>) and then twice 'Edit' 'Paste' (<Ctrl>+<V>). That gives you two more circles of the exact same size lying on top of the first one. You can move the circles by clicking on the circle and dragging it with pressed left mouse button. The desired position should, in our case, be in a vertical row in the left half of the editor window. Doubleclick on one of the other two circles in order to open the configuration dialog box again. Enter in the field Change Color of the corresponding circle the following variables:

for the middle circle: L1_yellow
for the lowest circle: L1-green

Now choose for the circles in the category **Color** and in the area **Alarm color** the corresponding color (yellow or green).

The TRAFFICSIGNAL case

Now enter the command **'Insert' 'Rectangle'**, and insert in the same way as the circle a rectangle which encloses the three circles. Once again choose as neutral a color as possible for the rectangle and give the command **'Extras' 'Send to back'** so that the circles are visible again.

If simulation mode[1] is not yet turned on, you can activate it with the command **'Online' 'Simulation'**. If you now start the simulation with the commands **'Online' 'Login'** and **'Online' 'Run'**, then you can observe the color change of the first traffic signal.

[1] The simulation mode is active if a check mark (✓) appears in front of the menu item "Simulation" in the 'Online' menu..

The second traffic signal

The simplest way to create the second traffic signal is to copy all of the elements of the first traffic signal. For this you select all elements of the first traffic signal and copy them (as before with the lights of the first traffic signal) with the commands '**Edit**' '**Copy**' and '**Edit**' '**Paste**'. You then only have to change the text "TRAFFICSIGNAL1" in the respective dialog boxes into "TRAFFICSIGNAL2", and the visualization of the second traffic signal is completed.

The ON switch

Insert a rectangle and award it, as described above, a colour for a traffic light of your choice and enter .ON at **Variables** for the **Change color**. Enter ”ON” in the input field for **Content** in the category Text.

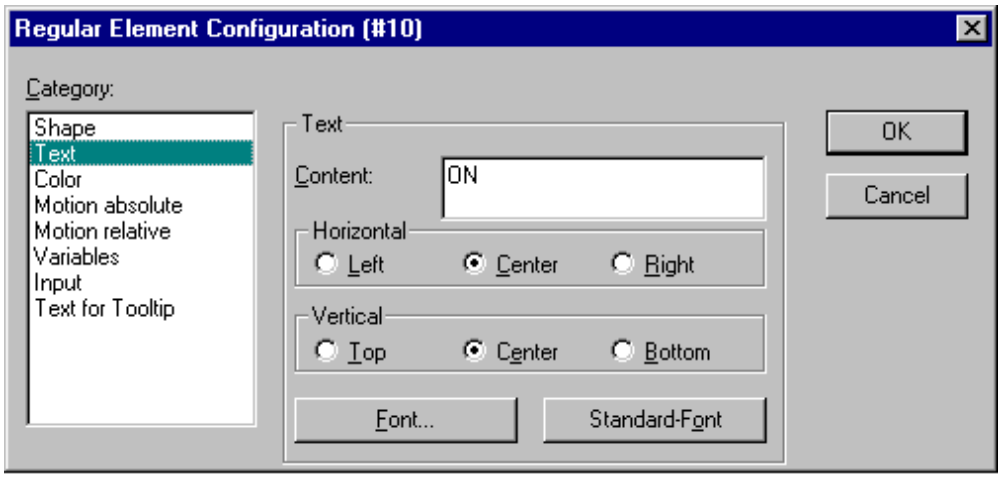


Image 3.23: Dialog to configure the visualization elements (Category Text)

In order to set the variable ON to TRUE with a mouse click on the switch, activate option 'Toggle variable' in category 'Input' and enter variable name ".ON" there. 'Tip Variable' means that when a mouse click is made on the visualization element the variable .ON is set to the value TRUE but is reset to the value FALSE as soon as the mousekey is released again (we have created hereby a simple switch-on device for our traffic lights program).

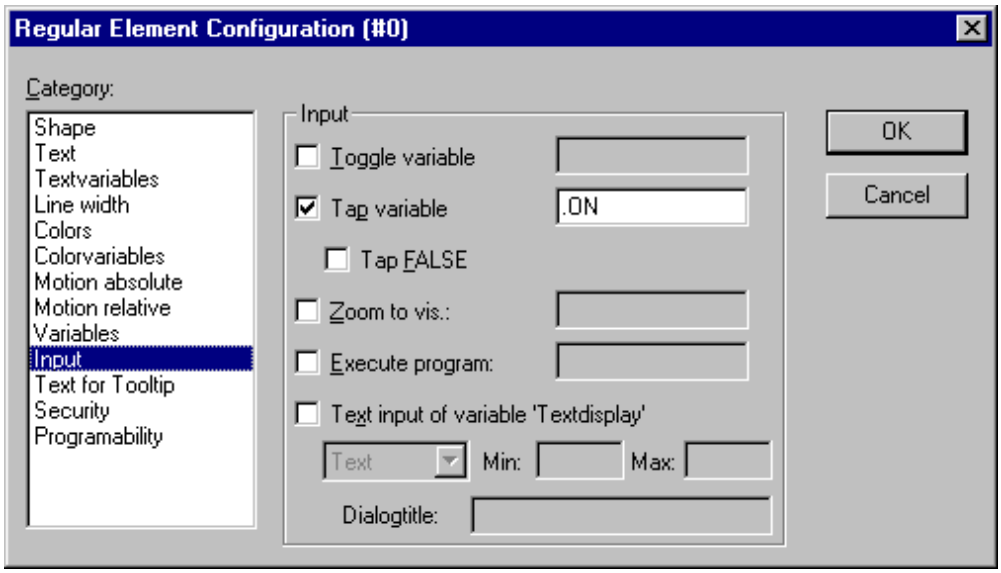


Image 3.24: Dialog to configure the visualization elements (Category Input)

Font in the visualization

In order to complete the visualization you should first insert two more rectangles which you place underneath the traffic signals.

In the visualizations dialog box set white in the category **Color** for **Frame** and write in the category **Text** in the field **Contents** "Light1" or "Light2". Now your visualization looks like this:

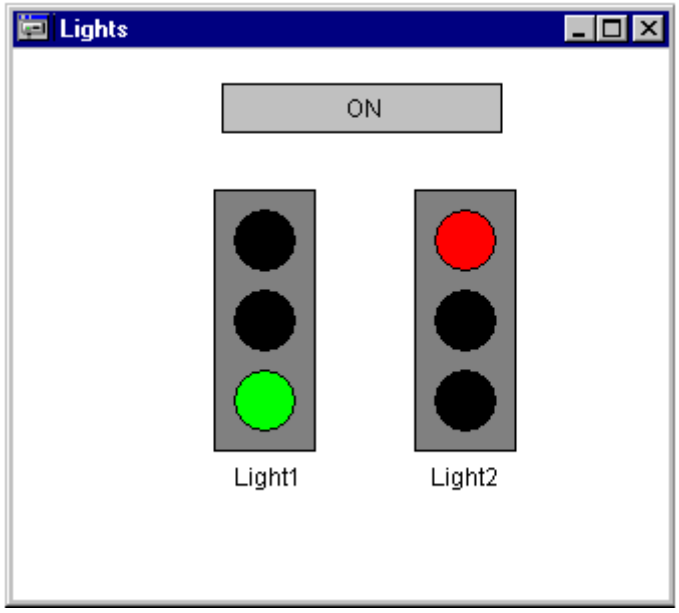


Image 3.25: Visualization for the Sample Project Trafficsignal

4 The Individual Components

4.1 The Main Window

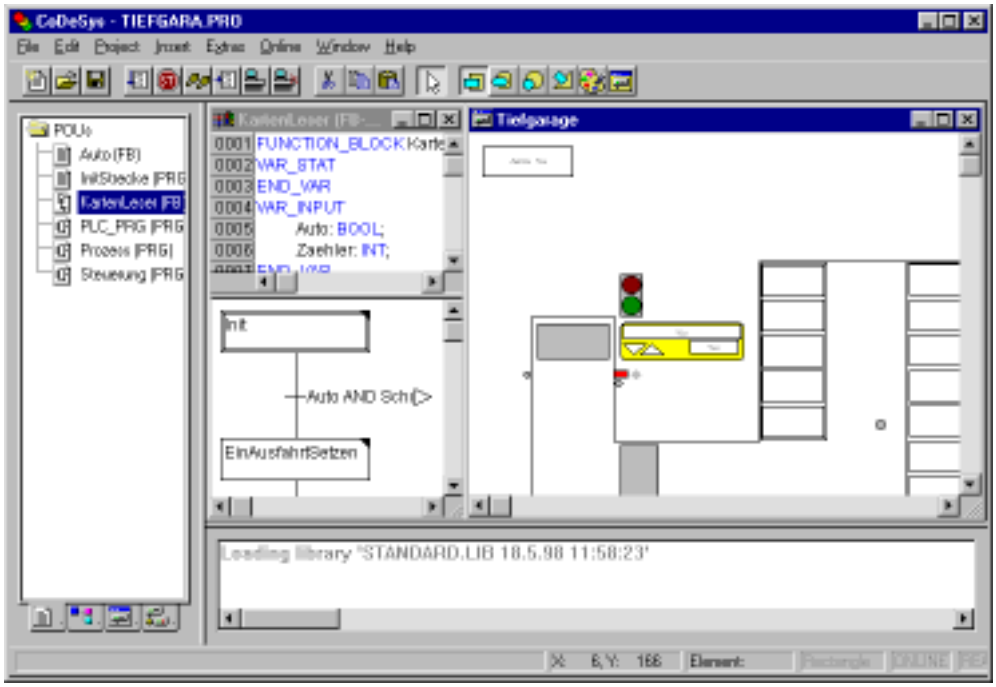


Image 4.1: The Main Window

The following elements are found in the main window of **CoDeSys** (from top to bottom):

- The **menu bar**
- The **Tool bar** (optional); with buttons for faster selection of menu commands.
- The **Object Organizer** with register cards for POUs, Data types, Visualizations, and Resources
- A vertical **screen divider** between the Object Organizer and the Work space of CoDeSys
- The **Work space** in which the editor windows are located
- The **message window** (optional)
- The **Status bar** (optional); with information about the current status of the project

Menu bar

The menu bar is located at the upper edge of the main window. It contains all menu commands.

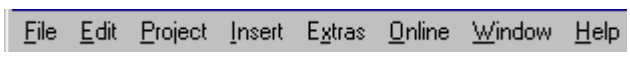


Image 4.2: Menu Bar

Tool bar

By clicking with the mouse on a symbol you can select a menu command more quickly. The choice of the available symbols automatically adapts itself to the active window.

The command is only carried out when the mouse button is pressed on the symbol and then released.

If you hold the mouse pointer for a short time on a symbol in the tool bar, then the name of the symbol is shown in a Tooltip.

In order to see a description of each symbol on the tool bar, select in Help the editor about which you want information and click on the tool bar symbol in which you are interested.

The display of the tool bar is optional (see Chapter 2.2, 'Project' 'Options' category Desktop).

Image 4.3: Tool bar with symbols

Object Organizer

The Object Organizer is always located on the left side of CoDeSys. At the bottom there are four register cards with symbols for the four types of objects **POUs**, **Data types**, **Visualizations** and **Resources.** In order to change between the respective object types click with the mouse on the corresponding register card or use the left or right arrow key.

You will learn in chapter Managing Objects how to work with the objects in the Object Organizer.

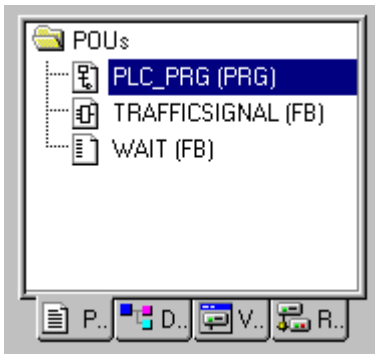


Image 4.4: Object Organizer

Screen divider

The screen divider is the border between two non-overlapping windows. In **CoDeSys** there are screen dividers between the Object Organizer and the Work space of the main window, between the interface (declaration part) and the implementation (instruction part) of POUs and between the Work space and the message window.

You can move the screen divider with the mouse pointer. You do this by moving the mouse with the left mouse button pressed.

Make sure the screen divider always remains at its absolute position, even when the window size has been changed. If it seems that the screen divider is no longer present, then simply enlarge your window.

Work space

The Work space is located on the right side of the main window in **CoDeSys**. All editors for objects and the library manager are opened in this area. The current object name appears in the title bar; in the case of POUs an abbreviation for the POU type and the programming language currently in use appears in brackets after it.

You find the description of the editors in Chapter 5 'Editors in CoDeSys'

Under the menu item **'Window'** you find all commands for window management.

Message window

The message window is separated by a screen divider underneath the work space in the main window.

It contains all messages from the previous compilations, checks or comparisons. Search results and the cross-reference list can also be output here.

If you doubleclick with the mouse in the message window on a message or press <Enter>, the editor opens with the object. The relevant line of the object is selected. With the commands **'Edit' 'Next error'** and **'Edit' 'Previous error'** you can can quickly jump between the error messages.

The display of the message window is optional (see 'Window' 'Messages').

Status bar

The status bar at the bottom of the window frame of the main window in **CoDeSys** gives you information about the current project and about menu commands.

If an item is relevant, then the concept appears on the right side of the status bar in black script, otherwise in gray script.

When you are working in online mode, the concept **Online** appears in black script. If you are working in the offline mode it appears in gray script.

In Online mode you can see from the status bar whether you are in the simulation (**SIM**), the program is being processed (**RUNS**), a breakpoint is set (**BP**), or variables are being forced (**FORCE**).

With text editor the line and column number of the current cursor position is indicated (e.g. **Line:5, Col.:11**). In online mode '**OV**' is indicated black in the status bar. Pressing the <Ins> key switches between Overwrite and Insert mode.

If the mouse point is in a visualization, the current **X** and **Y position** of the cursor in pixels relative to the upper left corner of the screen is given. If the mouse pointer is on an **Element**, or if an element is being processed, then its number is indicated. If you have an element to insert, then it also appears (e.g. **Rectangle**).

If you have chosen a menu command but haven't yet confirmed it, then a short description appears in the status bar.

The display of the statusbar is optional (see Chapter 2.2, 'Project' 'Options' category Desktop).

Context Menu

Shortcut: <Shift>+<F10>

Instead of using the menu bar for executing a command, you can use the right mouse button. The menu which then appears contains the most frequently used commands for a selected object or for the active editor. The choice of the available commands adapts itself automatically to the active window. The choice of the available commands adapts itself automatically to the active window.

4.2 Options

About **CoDeSys** there can be of course only one viewpoint. In **CoDeSys**, however, you can configure the view of the main window (and have more than one viewpoint). In addition you can make other settings. For this you have the command **'Project' 'Options'** at your disposal. The settings you make thereby are, unless determined otherwise, saved in the file "CoDeSys.ini" and restored at the next **CoDeSys** startup.

'Project' 'Options'

With this command the dialog box for setting options is opened. The options are divided into different categories. Choose the desired category on the left side of the dialog box by means of a mouse click or using the arrow keys and change the options on the right side.

You have at your disposal the following categories:

- Load & Save
- User information
- Editor
- Desktop
- Color
- Directories
- Log
- Build
- Passwords
- Sourcedownload
- Symbol configuration
- Project Source Control
- Macros

Load & Save

If you choose this category in the Options dialog box, you will get the following dialog box:

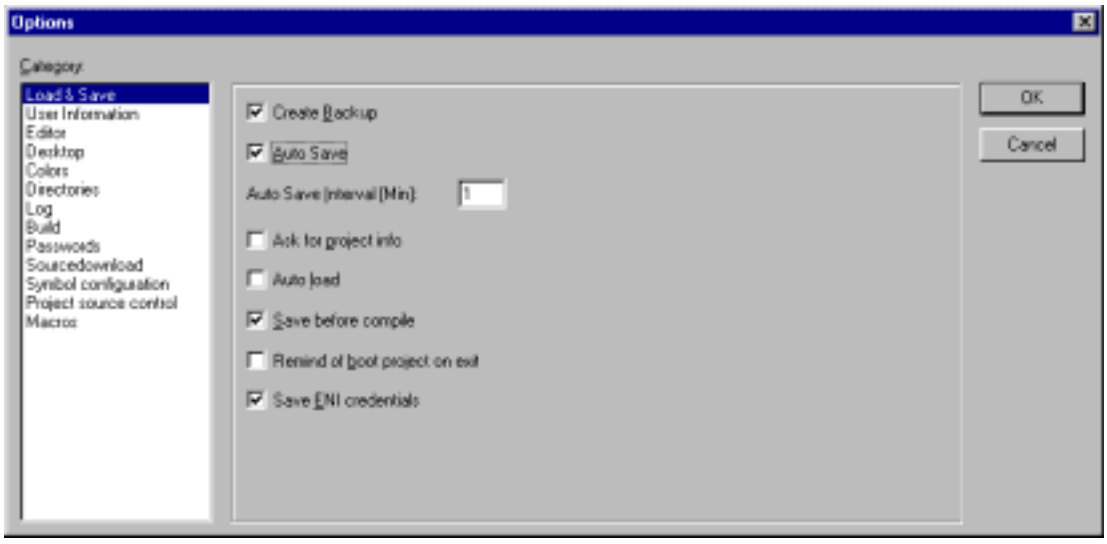


Image 4.5: Option dialog box of the category Load & Save

When activating an option, a check (✓) appears before the option.

Create Backup: **CoDeSys** creates a backup file at every save with the extension ".bak". Contrary to the *.asd-file (see below, 'Auto Save') this *.bak-file is kept after closing the project. So you can restore the version you had before the last project save.

Auto Save : While you are working your project is consecutively saved to a temporary file with the extension ".asd" according to a set time interval (**Auto Save Interval**). This file is erased at a normal exit from the program. If for any reason **CoDeSys** is not shut down "normally" (e.g. due to a power failure), then the file will not be erased. When you open the file again the following message appears:

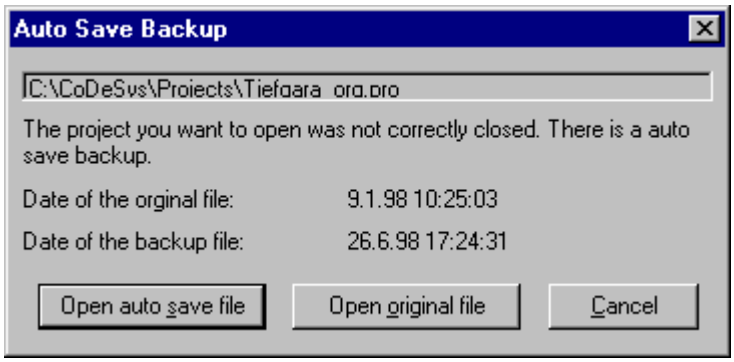

Auto Save Backup
C:\CoDeSys\Projects\Tiefgara_org.pro
The project you want to open was not correctly closed. There is a auto save backup.
Date of the orginal file: 9.1.98 10:25:03
Date of the backup file: 26.6.98 17:24:31
Open auto save file
Open original file
Cancel

Image 4.6: There is an auto save backup.

You can now decide whether you want to open the original file or the auto save file.

Ask for project info: When saving a new project or saving a project under a new name, the project info is automatically called. You can visualize the project info with the command '**Project**' '**Project info**' and also process it.

Auto Load: At the next start of **CoDeSys** the last open project is automatically loaded. The loading of a project at the start of **CoDeSys** can also take place by entering the project in the command line.

Save before compile: The project will be saved before each compilation. In doing so a file with the extension "**.asd**" will be created, which behaves like described above for the option 'Auto Save'.

Remind of boot project on exit: If the project has been modified and downloaded without creating a new boot project since the last download of a boot project, then a dialog will advise the user before leaving the project: "No boot project created since last download. Exit anyway ?".

Save ENI credentials: User name and Password, as they might be inserted in the Login dialog for the ENI data base, will be saved with the project.

User information

If you choose this category in the Options dialog box, then you get the following dialog box:

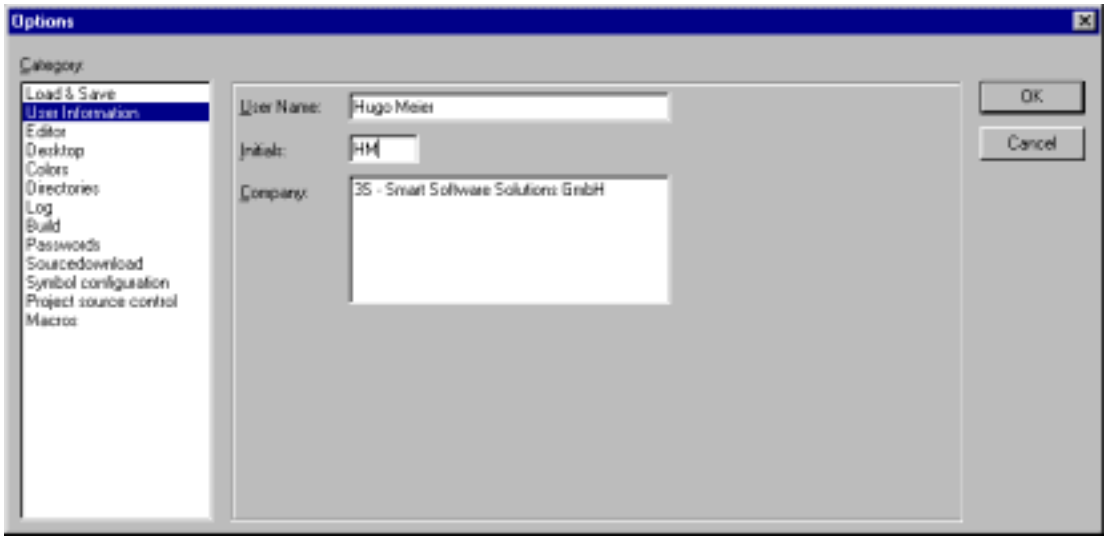


Image 4.7: Options dialog box of the category User information

To User information belong the **Name** of the user, his **Initials** and the **Company** for which he works. Each of the entries can be modified.

Editor

If you choose this category in the Options dialog box, then you get the following dialog box:

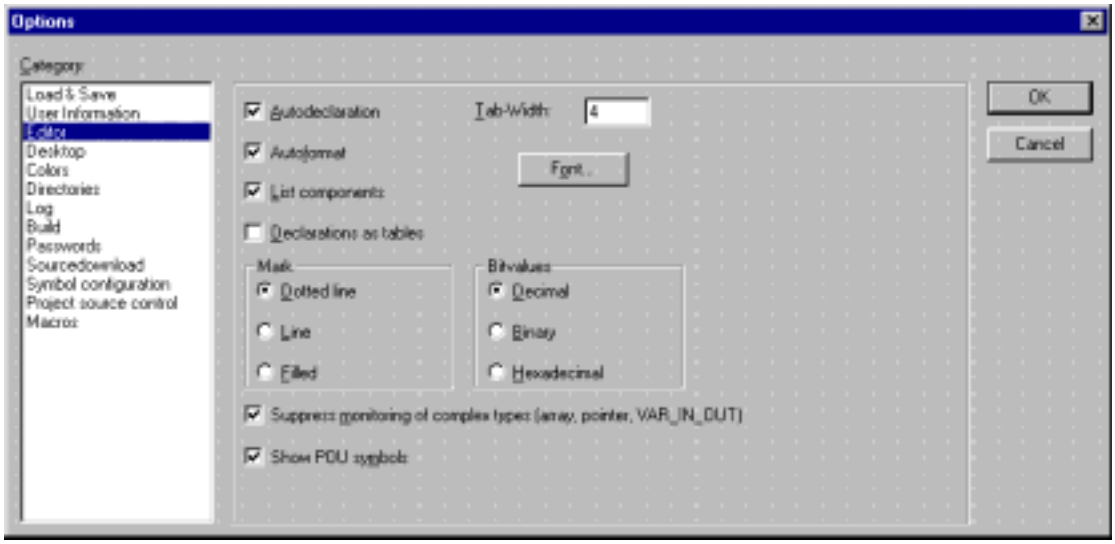


Image 4.8: Options dialog box of the category Editor

When activating an option, a check (✓) appears before the option.

You can make the following settings for the Editors:

Autodeclaration: If this option is activted, then after the input of a not-yet-declared variable a dialog box will appear in all editors with which this variable can be declared.

Autoformat: If this option is activated, then **CoDeSys** will execute automatic formatting in the IL editor and in the declaration editor. When you finish with a line, the following formatting is made:

- Operators written in small letters are shown in capitals;
- Tabs are inserted to that the columns are uniformly divided.

List components: If this option is activated, then the **Intellisense** functionality will be available to work as an input assistant. This means that if you insert a dot at a position where a identifier should be inserted, then a selection list will open, offering all global variables which are found in the project. If you insert the name of a function block instance, then you will get a selection list of all inputs and outputs of the instanced function block. The Intellisense function is available in editors, in the Watch- and Receiptmanager, in visualizations and in the Sampling Trace.

Declarations as tables: If this option is activated, then you can edit variables in a table instead of using the usual declaration editor. This table is sorted like a card box, where you find tabs for input variables, output variables local variables and in_out variables. For each variable there are edit fields to insert **Name, Address**, **Type**, **Initial** and **Commment**.

Tab-Width: In the field **Tab-Width** in the category **Editor** of the Options dialog box you can determine the width of a tab as shown in the editors. The default setting is four characters, whereby the character width depends upon the font which is chosen.

Font: By clicking on the button **Font** in the category **Editor** of the Options dialog box you can choose the font in all **CoDeSys** editors. The font size is the basic unit for all drawing operations. The choice of a larger font size thus enlarges the printout,

even with each editor of **CoDeSys**. After you have entered the command, the font dialog box opens for choosing the font, style and font size.

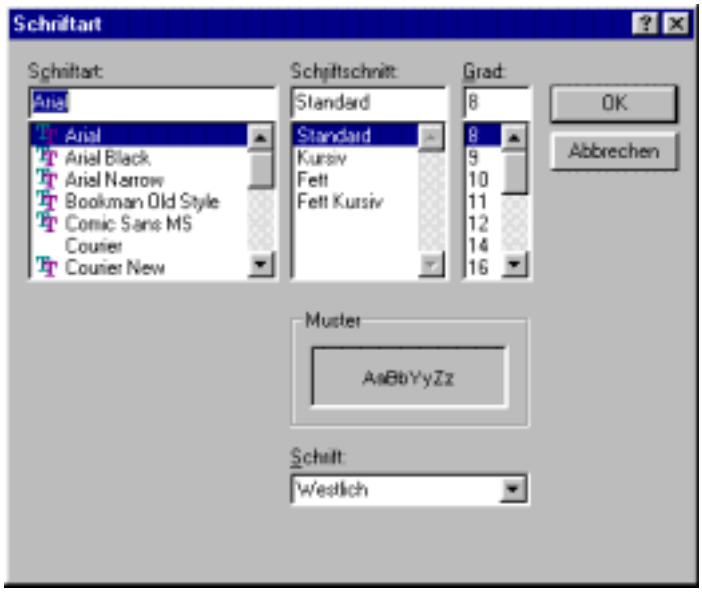


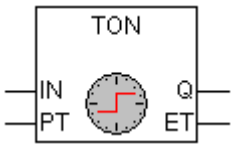
Image 4.9: Dialog box for setting the font

Mark: When choosing **Mark** in the **Editor** category in the Options dialog box you can choose whether the current selection in your graphic editors should be represented by a dotted rectangle (**Dotted**), a rectangle with continuous lines (**Line**) or by a filled-in rectangle (**Filled**). In the last case the selection is shown inverted.

Bitvalues: When choosing **Bitvalues** in the category **Editor** of the Options dialog box you can choose whether binary data (type BYTE, WORD, DWORD) during monitoring should be shown **Decimal**, **Hexadecimal**, or **Binary**.

Suppress monitoring of complex types (array, ointer, VAR_IN_OUT): If this option is activated, then complex data types like arrays, pointers or VAR_IN_OUTs will not be displayed in the monitoring window in online mode.

Show POU symbols: If this option is activated, then symbols will be displayed in POU boxes, provided that those are available as bitmaps in the library folder. The name of the bitmap file must be composed of the POU name and the extension ".bmp". Example: The symbol file for POU TON must be named TON.bmp:



The selection is activated in front of which a (•) point appears.

Desktop

If you choose this category in the Options dialog box, then you get the following dialog box:

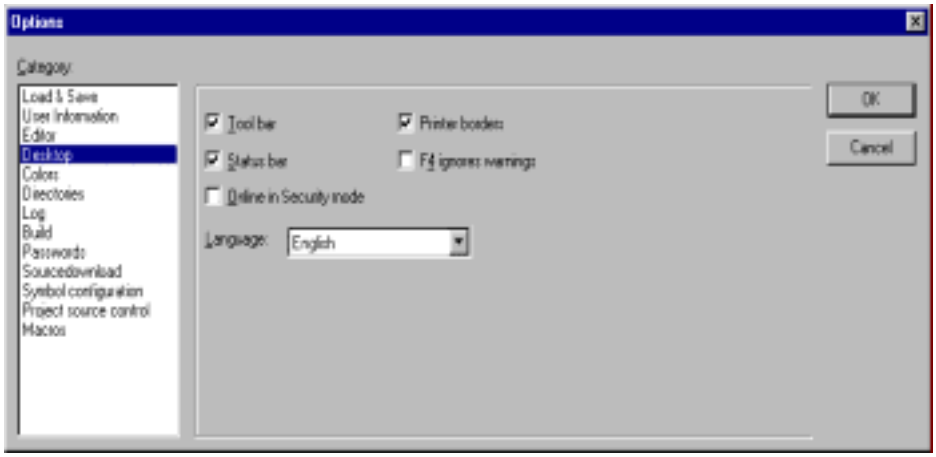


Image 4.10: Options dialog box of the category Desktop

Tool bar: The tool bar with the buttons for faster selection of menu commands becomes visible underneath the menu bar.

Status bar: The status bar at the lower edge of the **CoDeSys** main window becomes visible.

Online in Security mode: In Online mode with the commands '**Run**', '**Stop**', '**Reset**' '**Toggle Breakpoint**', '**Single cycle**', '**Write values**', '**Force values**' and '**Release force**', a dialog box appears with the confirmation request whether the command should really be executed. This option is saved with the project.

Printer borders: In every editor window, the limits of the currently set print range are marked with red dashed lines. Their size depends on the printer characteristics (paper size, orientation) and on the size of the "Content" field of the set print layout (menu: 'File" "Documentation Settings").

F4 ignores warnings: After compilation, when F4 is pressed in a message window, the focus jumps only to lines with error messages; warning messages are ignored.

Language: Define here, in which language the menu and dialog texts should be displayed.

Note: Please note, that the language choice is only possible under Windows NT!

When an option is activated, a check appears in front of it.

Colors

If you choose this category in the Options dialog box , then you get the following dialog box:

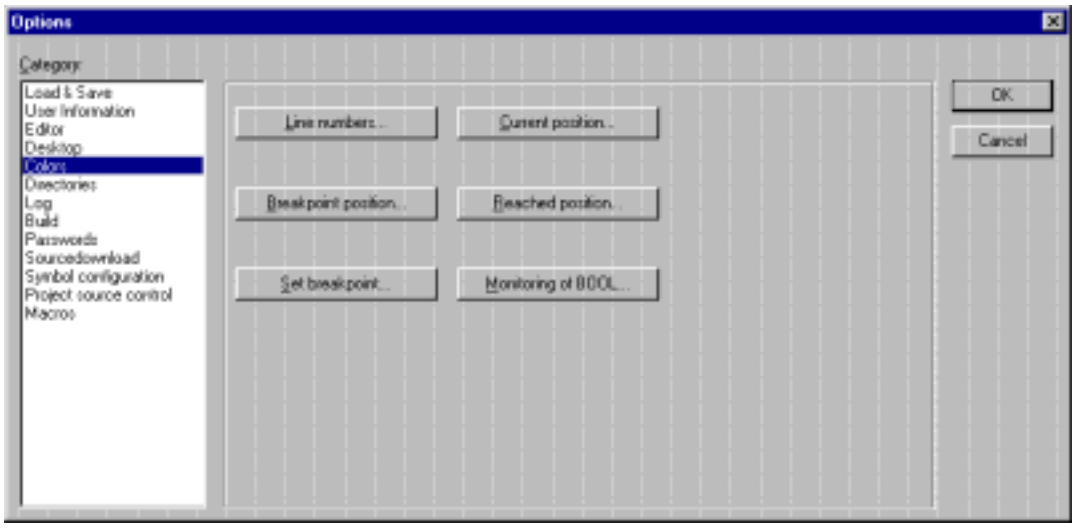


Image 4.11: Options dialog box of the category Color

You can edit the default color setting of **CoDeSys**. You can choose whether you want to change the color settings for **Line numbers** (default presetting: light gray), for **Breakpoint positions** (dark gray), for a **Set breakpoint** (light blue), for the **Current position** (red), for the **Reached Positions** (green) or for the **Monitoring of Bool**ean values (blue).

If you have chosen one of the indicated buttons, the dialog box for the input of colors opens.

Image 4.12: Dialog box for setting colors

Directories

If you choose this category in the Options dialog box, then you get the following dialog box:

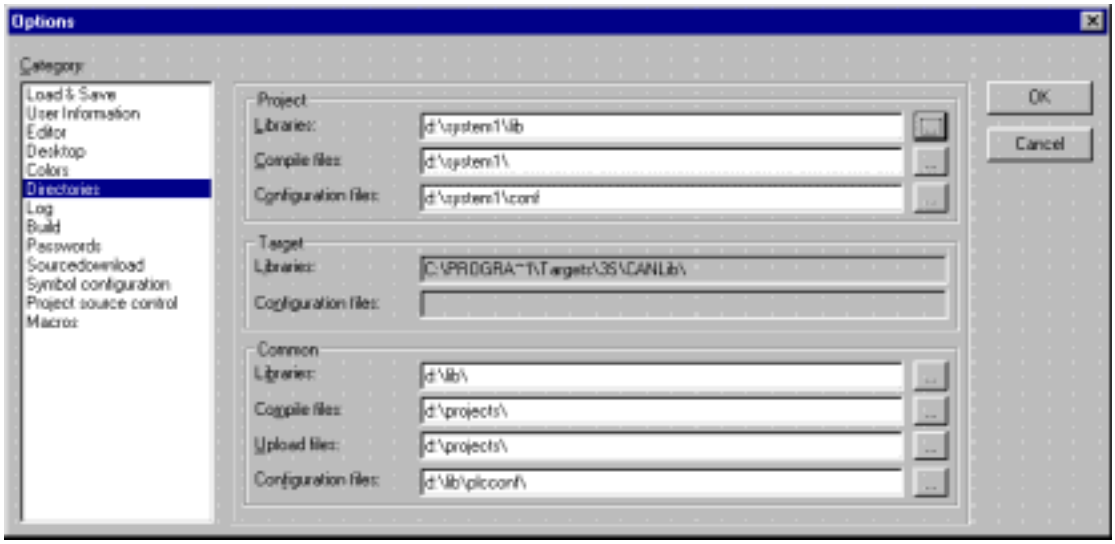


Image 4.13: Options dialog box of the category Directories

Directories can be entered in the **Project** and **Common** areas for **CoDeSys** to use in searching for **libraries** and controller **configuration** files, as well as for storing **compile** and source-**upload files**. If you activate the button (...) behind a field, the directory selection dialog opens. For library and configuration files, several paths can be entered for each, separated by semicolons ";".

The information in the **Project** area is stored with the project; information in the **Common** area is written to the ini file of the programming system and thus apply to all projects.

The **Target** area displays the directories for libraries and configuration files set in the target system, e.g. through entries in the Target file. These fields cannot be edited, but an entry can be selected and copied (right mouse button context menu).

CoDeSys generally searches first in the directories entered in 'Project', then in those in 'Target System' (defined in the Target file), and finally those listed under 'Common'. If two files with the same name are found, the one in the directory that is searched first will be used.

Options for Log

If you choose this category in the Options dialog box, then you get the following dialog box shown below. In this dialog, you can configure a file that acts as a project log, recording all user actions and internal processes during Online mode processing (see in this connection Chapter 2.7, Log).

If an existing project is opened for which no log has yet been generated, a dialog box opens which calls attention to the fact that a log is now being set up that will receive its first input after the next login process.

The log is automatically stored as a binary file in the project directory when the project is saved. If you prefer a different target directory, you can activate the option **Directory for project logs:** and enter the appropriate path in the edit field. Use the button to access the "Select Directory" dialog for this purpose.

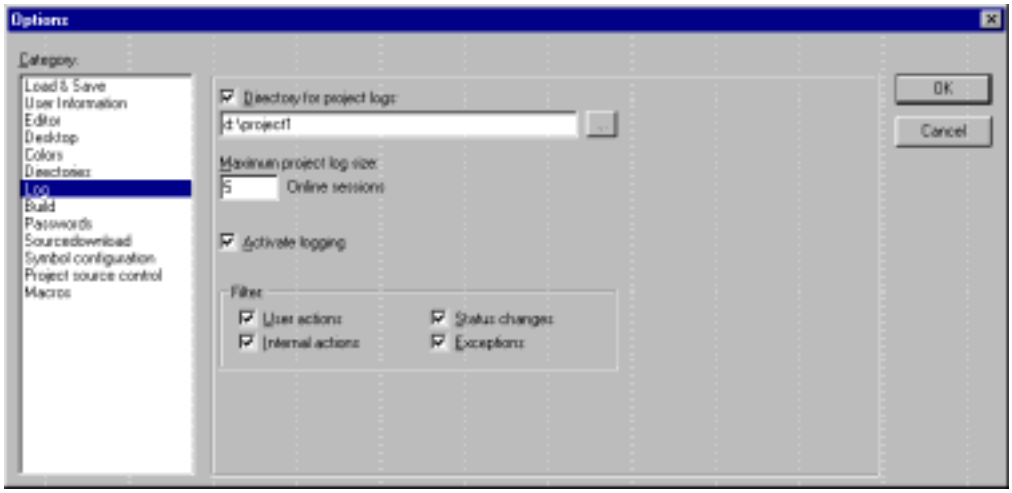


Image4.14: Option dialogue for the category Log

The log file is automatically assigned the name of the project with the extension .log. The maximum number of **Online sessions** to be recorded is determined by **Maximum project log size**. If this number is exceeded while recording, the oldest entry is deleted to make room for the newest.

The Log function can be switched on or off in the Option field **Activate logging**.

You can select in the **Filter** area which actions are to be recorded: User actions, Internal actions, Status changes, Exceptions. Only actions belonging to categories checked here will appear in the Log window and be written to the Log file. (For a description of the categories, please see chapter 2.7, Log).

The Log window can be opened with the command 'Window' 'Log' (see also Chapter 2.7, Log).

Build

If you choose this category in the Options dialog box, you will get the following dialog box:

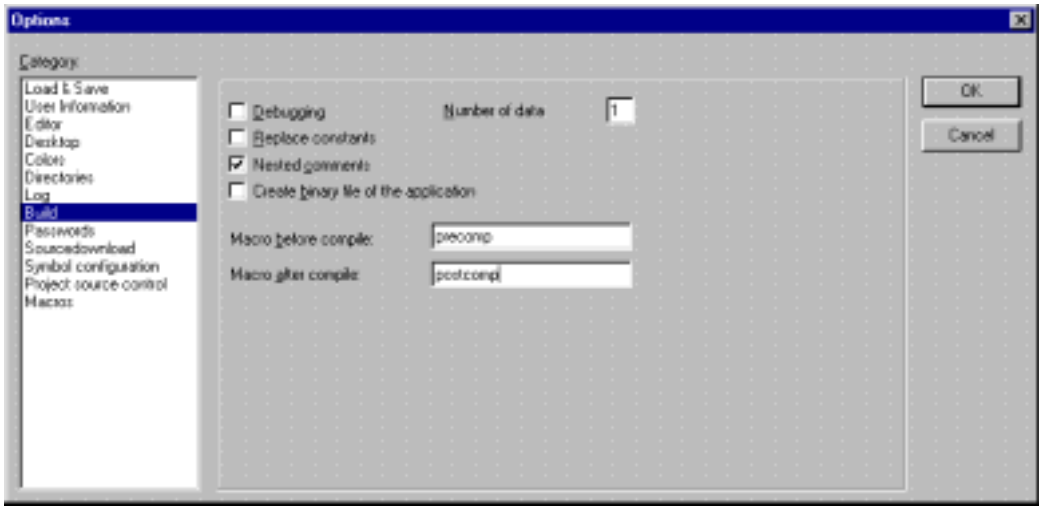


Image 4.15: Options dialog box of the category Build

Debugging: Additional debugging code is created, that is the code can become considerably larger. The debugging code is needed in order to make use of the

debugging functions offered by **CoDeSys** (e.g. breakpoints). When you switch off this option, project processing becomes faster and the size of the code decreases. The option is stored with the project.

Replace constants: The value of each constant is loaded directly, and in Online mode the constants are displayed in green. Forcing, writing and monitoring of a constant is then no longer possible. If the option is deactivated, the value is loaded into a storage location via variable access (this does in fact allow writing the variable value, but implies longer processing time).

Nested comments: Comments can be placed within other comments. Example:

```
(*
a:=inst.out; (* to be checked *)
b:=b+1;
*)
```

Here, the comment that begins with the first bracket is not closed by the bracket following “checked,” but only by the last bracket.

Create binary file of the application: A binary image of the generated code (boot project) is created in the project directory during compilation. File name: <project_name>.bin. By comparison, the command 'Online' 'Create Boot project' sets up the boot project on the controller.

Number of data Enter here how many storage segments are to be reserved for your project data in the controller. This space is required so that an Online Change can still be carried out when new variables are added. When compiling the project you might get the message "The global variables require too much memory. Increase the available memory in the project options."

In order to exert control over the compilation process you can set up two macros: the macro in the **Macro before compile** field is executed before the compilation process; the macro in the **Macro after compile** field afterwards. The following macro commands can not, however, be used here: file new, file open, file close, file save as, file quit, online, project compile, project check, project build, debug, watchlist.

All entries in the Build Options dialog are stored with the project.

When an option is activated, a check appears in front of it.

Passwords

If you choose this category in the Options dialog box, then you get the dialog box shown below. To protect your files from unauthorized access **CoDeSys** offers the option of using a password to protect against your files being opened or changed.

Enter the desired password in the field **Password**. For each typed character an asterisk (*) appears in the field. You must repeat the same word in the field **Confirm Password**. Close the dialog box with **OK**. If you get the message:

"The password does not agree with the confirmation",

then you made a typing error during one of the two entries. In this case repeat both entries until the dialog box closes without a message.

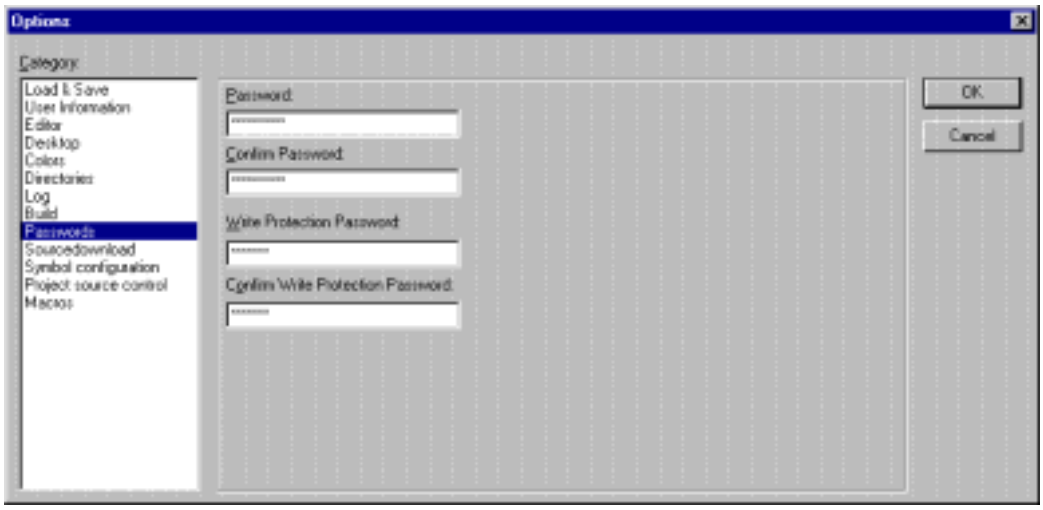


Image 4.16: Options dialog box of the category Passwords

If you now save the file and then reopen it, then you get a dialog box in which you are requested to enter the password. The project can then only be opened if you enter the correct password. Otherwise **CoDeSys** reports:

"The password is not correct."

Along with the opening of the file, you can also use a password to protect against the file being changed. For this you must enter a password in the field **Write Protection Password** and confirm this entry in the field underneath.

A write-protected project can be opened without a password. For this simply press the button **Cancel**, if **CoDeSys** tells you to enter the write-protection password when opening a file. Now you can compile the project, load it into the PLC, simulate, etc., but you cannot change it.

Of course it is important that you memorize both passwords. However, if you should ever forget a password, then contact the manufacturer of your PLC.

The passwords are saved with the project.

In order to create differentiated access rights you can define user groups and "Passwords for user groups").

Sourcedownload

When you select this category, the dialog shown below will be opened

You can choose to which **Timing** and what **Extent** the project is loaded into the controller system. The option **Sourcecode only** exclusively involves just the CoDeSys file (file extension .pro). The option **All files** also includes files such as the associated library files, visualization bitmaps, configuration files, etc.

Using the option **Implicit at load** allows the selected file range to be automatically loaded into the controller system on the command **‘Online‘ ‘Download'**.

Using the option **Notice at load** offers a dialog, when the command **‘Online‘ ‘Download'** is given, with the question ”Do you want to write the source code into the controller system?”. Pressing **Yes** will automatically load the selected range of files into the controller system, or you can alternatively finish with **No**.

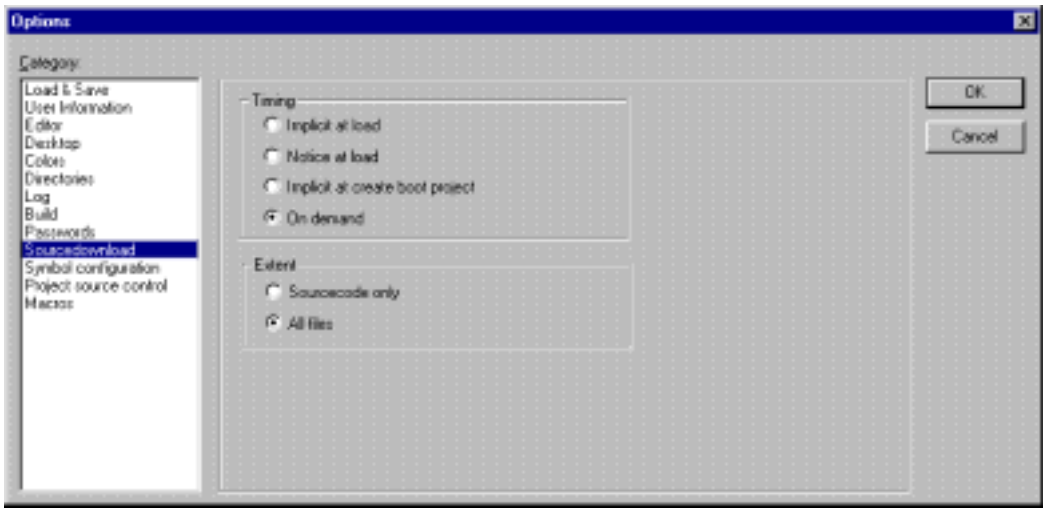


Image 4.17: Option dialog for the category Sourcedownload

When using the option **On demand** the selected range of files must be expressly loaded into the controller system by giving the command **'Online' 'Sourcecode download'**.

The project which is stored in the controller system can be retrieved by using **'File' 'Open'** with **Open project from PLC**. The files will be unpacked in the process. See Chapter 2.3, 'File' 'Open' for details !

Options for Symbol Configuration

The dialog presented here is used for configuring the symbol file (text file *.sym and binary file *.sdb). This will be created as a text file <project name>.sym resp. a binary file <project name>.sdb (depending on the used gateway version) in the project directory. The symbol file is needed for data exchange with the controller via the symbol interface and are used for that purpose e.g. by OPC- or GatewayDDE-Server.

If the option **Create symbol entries** is selected, then symbol entries for the project variables will be automatically created in a symbol file at each compilation of the project.

If additionally the option **Dump XML symbol table** is activated, then also an XML file containing the symbol information will be created in the project directory. It will be named <project name>.SYM_XML.

Regard the following when configuring the symbol entries:

- If option 'Symbol config from INI-file' is activated in the target settings, then the symbol configuration will be read from the codesys.ini or another ini-file, which is defined there. (In this case the dialog 'Set object attributes' in CoDeSys is not editable).
- If option 'Symbol config from INI-file' is not activated, the symbol entries will be generated in accordance with the settings you make in the 'Set object attributes' dialog. You get there using the **Configure symbol file** button:

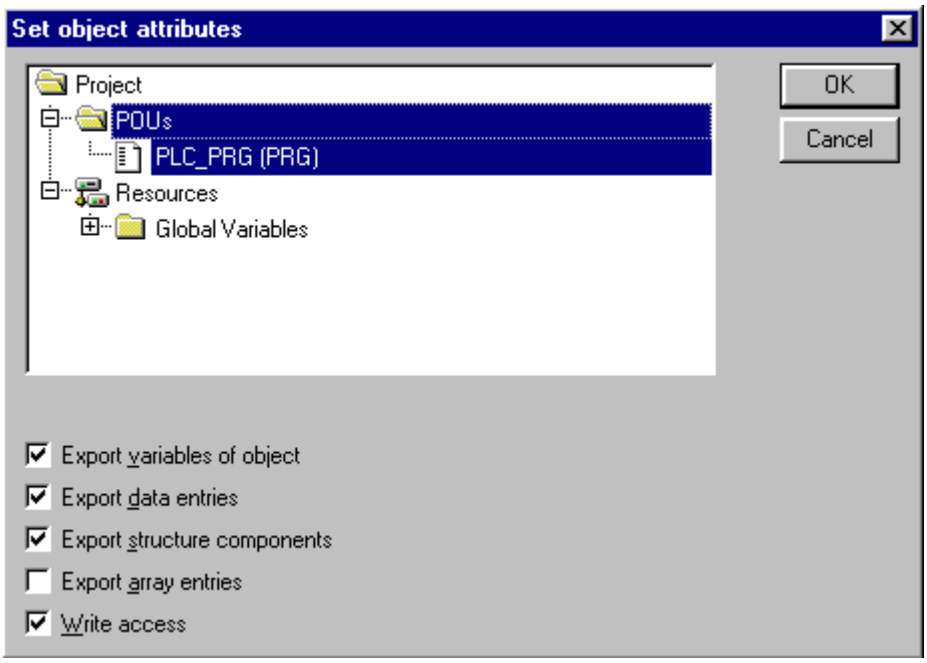


Image4.18: Set object attributes dialog (in option category Symbol configuration)

Use the tree-structured selection editor to select project POUs and set the desired options in the lower part of the dialog box by clicking the mouse on the corresponding small boxes. Activated options are checked. The following options can be set:

Export variables of object: The variables of the selected object are exported in the symbol file.

The following options can take effect only if the **Export variables of object** option is activated:

Export data entries: Entries for access to the global variables are created for object's structures and arrays.

Export structure components: An individual entry is created for each variable component of object's structures.

Export array entries: An individual entry is created for each variable component of object's arrays.

Write Access: Object's variables may be changed by the OPC server.

Once the option settings for the currently selected POU are complete, other POUs can also be selected - without closing the dialog before - and can be given an option configuration. This can be carried out for any desired number of POU selections, one after the other. When the dialog box is closed by selecting **OK**, all configurations carried out since the dialog box was opened are applied.

Options for 'Project source control'

This dialog is used to define whether the project should be managed in a project data base and to configure the ENI interface correspondingly.

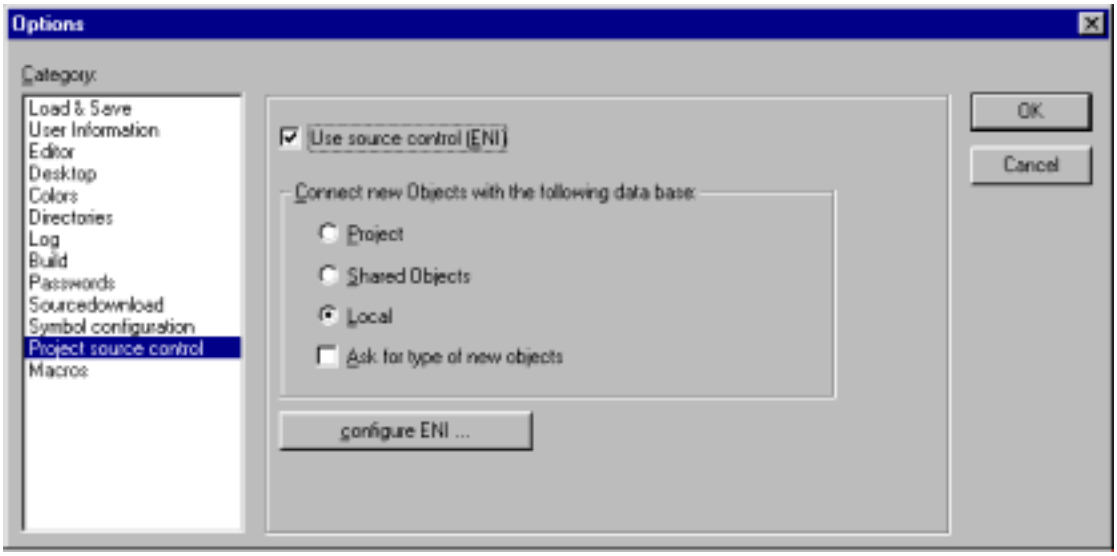


Image4.19: Options dialog box of category Project source control

Use source control (ENI): Activate this option, if you want to access a project data base via the ENI Server in order to administer all or a selection of POUs of the project in this data base. Preconditions: ENI Server and data base must be installed and you must be registered as user of both. See also the separate documentation for the ENI-Server and see Chapter 7, 'The CoDeSys ENI'.

If the option is activated, then the data base functions (Check in, Get last version etc.) will be available for handling the project POUs. Then some of the data base functions will run automatically like defined in the options dialogs, and in the menu 'Project' 'Data Base Link' you will find the commands for calling the functions explicitely. Besides that a tab 'Data base-connection' will be added in the dialog 'Properties', where you can assign a POU to a particular data base category (see Chapter 7.3).

Connect new Objects with the following data base:

Here you set a default: When a new object is inserted in the project ('Project' 'Object' 'Add'), then it gets automatically assigned to that object category which is defined here. This assignment will be displayed in the object properties dialog ('Project' 'Object' 'Properties') and can be modified there later. The possible assignments:

> **Project:** The POU will be stored in that data base folder which is defined in the dialog 'ENI configuration/Project objects' in the field 'Project name'.
>
> **Shared Objects:** The POU will be stored in that data base folder which is defined in the dialog 'ENI configuration/Shared objects' (field 'Project name').
>
> **Local:** The POU will not be managed in a ENI data base, but only will be stored locally in the project.

Besides 'Project objects' and 'Shared objects' there is a third data base category 'Compile files' for such objects which are not created until the project has been compiled. Therefore this category is not relevant for the current settings.

Ask for type of new objects: If this option is activated, then whenever a new object is added to the project, the dialog 'Object' 'Properties' will open, where you can

choose to which of the three object categories mentioned above the POU should be assigned. By doing so the standard setting can be overwritten.

configure ENI: This button opens the first of three ENI configuration dialogs:

Each object of a project, which is determined to get managed in the ENI data base, can be assigned to one of the following data base categories: 'Project objects', 'Shared objects' or 'Compile files'. For each of these categories a separate dialog is available to define in which data base folder it should be stored and which presettings should be effective for certain data base functions:

- Dialog ENI configuration / Project objects
- Dialog ENI configuration / Shared objects
- Dialog ENI configuration / Compile files

Note: Each object will be stored also locally (with project) in any case.

The dialog will open one after the other if you are doing a primary configuration. In this case a **Wizard** (Button **Next**) will guide you and the settings entered in the first dialog will be automatically copied to the other ones, so that you just have to modify them if you need different parameter values.

If you want to modify an existing configuration, then the three dialogs are combined in one window (three tabs).

If you have not yet logged in successfully to the data base before, then the **Login** dialog will be opened automatically.

Options for project objects and shared objects regarding the project data base

These dialogs are part of the configuration of the project data base options ('Project' 'Options' 'Project source control'). Here you define the access parameters for the data base categories 'Project objects' und 'Shared objects'. Both dialogs contain the same items. (A third dialog is available for the configuration of the access to the data base category 'Compile files'.)

ENI-Connection

TCP/IP-Address:	Address of the computer where the ENI-Server is running
Port:	Default: 80; must be the same as set in the configuration parameters of the ENI Server
Project name:	Name of the data base folderwhere the objects of this category should be stored. Press button ... to open a folder tree of the already existing data base projects. If the desired folder already exists, you can select it in this tree and its name will be entered in the 'Project name' edit field. If you had not logged in to the ENI Server until you try to open the folder tree by button ..., then you will first get the **Login** dialog where you must enter 'User name' and 'Password' as defined in your ENI user account to get access to the three data base categories.

Read only If this option is activated, then only read access is possible to the above defined data base folder.

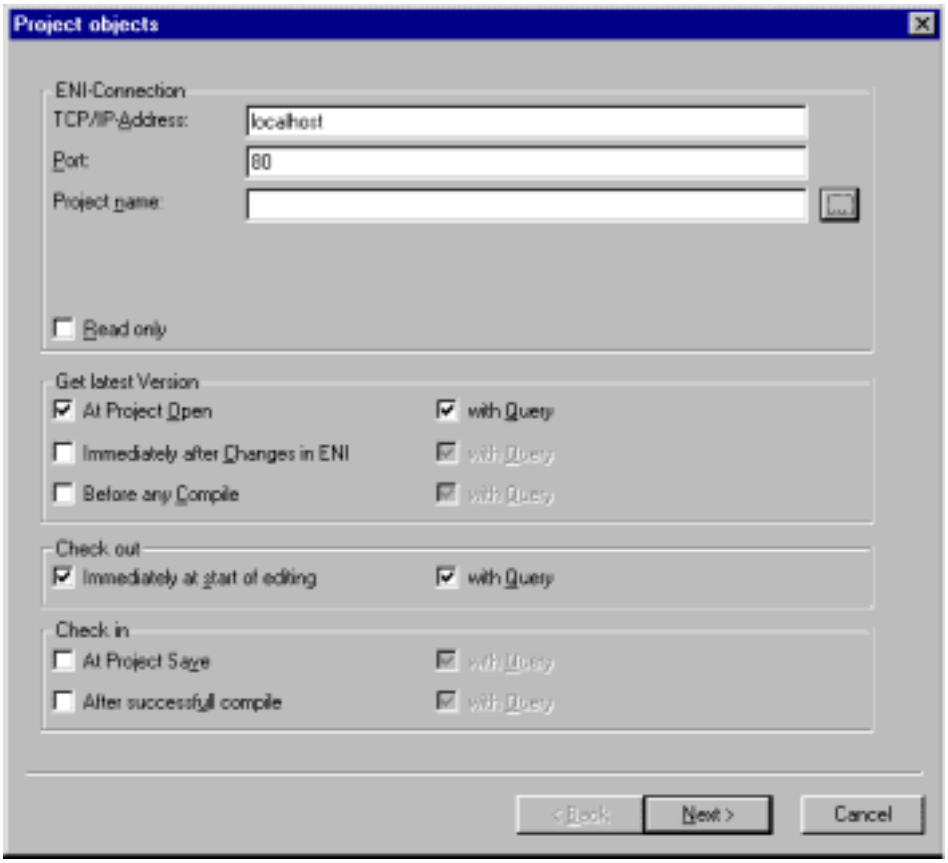


Image4.20: Dialog 'Project objects' in options category Project source control

Get latest Version

The data base function 'Get latest Version' (Menu 'Project' 'Data Base Link') copies the latest version of POUs from the above defined data base folder to the currently opened project, whereby the local version of objects will be overwritten. This will be done automatically for all objects, for which the version found in the data base differs from that in the project, as soon as one of the set timing conditions will meet. Activate the desired time options by setting a check mark:

At Project Open	As soon as the project is opened in CoDeSys
Immediately after Changes in ENI	As soon as a newer version of the POU is checked in to the data base (e.g. by another user); then the POU will be updated in the current project immediately and an appropriate message will pop up.
Before any Compile	Before any compile process in CoDeSys

Check out

The data base function 'Check out' means that the POU will be marked as 'in the works' and will be locked for other users until it will be de-blocked again by a 'Check in' or 'Undo check out' command.

If the option **Immediately at start of editing** is activated, then an object will be checked out automatically as soon as you start to edit it. If the object is currently already checked out by another user (indicated by a red cross before the object name in the CoDeSys object organizer), then a message will pop up.

Check in

The data base function 'Check in' means, that a new version of the object will be created in the data base. The older versions will be kept anyway.

You can activate one or both of the following options to define the time of automatic Checking in:

At Project Save	as soon as the project is saved
After successfull compile	as soon as the project has been compiled without errors

For each of the options 'Get last version', 'Check out' and 'Check in' additionally the option **with Query** can be activated. In this case, before the corresponding action is carried out, a dialog opens where you still can decide to cancel the action or otherwise confirm it.

The items of the dialog 'Shared objects' are the same like in the dialog 'Project objects' described above. The settings apply to all objects which are assigned to the data base category 'Shared objects'.

If you do a primary configuration, the configuration dialogs will appear one after the other and you will be guided by a wizard (button **Next**). The settings made in the first dialog will automatically be inherited to the other ones. So those just have to be edited if modificiations are necessary.

Cancel will close the dialog without saving the done modifications in the currently opened dialog. You return to the main dialog 'Options' 'Project source control'.

If an already existing configuration has been modified, then the new settings (for all three dialogs) can be saved by pressing **OK**. After that the dialog will be closed and you return to the main dialog 'Options' 'Project source control'.

Options for Compile Files
regarding the project data base

This dialog is part of the option settings for the project data base ('Project' 'Options' 'Project source control'). Here you define how the objects of category 'Compile files' will be handled in the data base. (Besides that two further dialogs are available to define this for objects of category 'Project objects' and 'Shared objects'.)

For the input fields **TCP/IP-Address**, **Port**, **Project name** see the description of dialog 'Project objects/Shared objects'.

Create ASCII-symbol information (.sym)	If this option is activated, then whenever a symbol file *.sym (text format) resp. *.sdb (binary format) will be created, this file will be written to the data base automatically. The entries in the symbol file are created like defined in the Project options category 'Symbol configuration'.
Create binary symbol information (.sdb)	
Create boot project	If this option is activated, then whenever a boot project

will be created, this file will be written to the data base automatically .

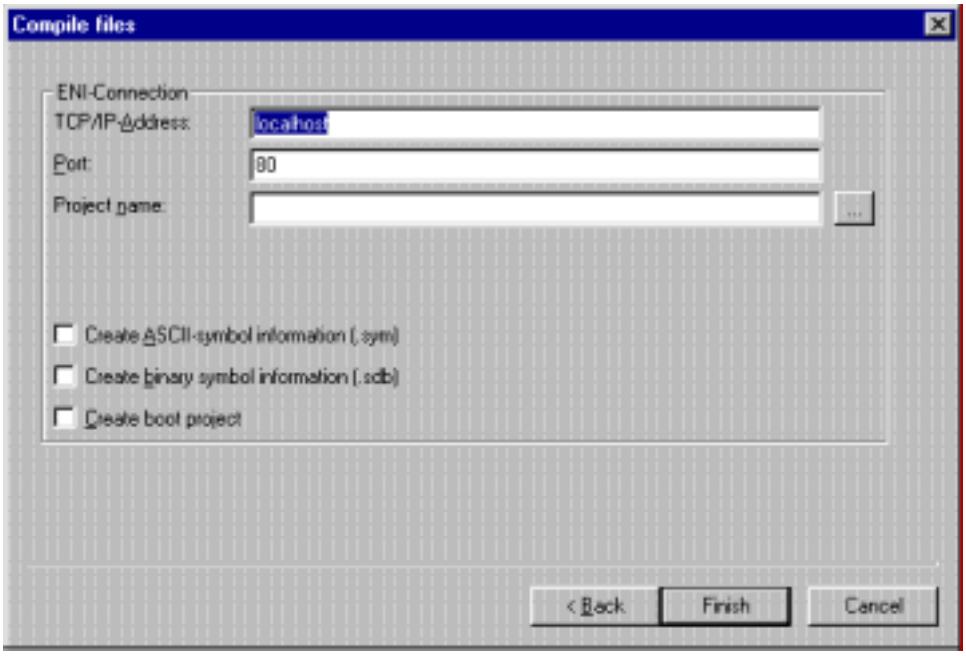


Image4.21: Dialog 'Compile files' in category Project source control

If you are doing a primary configuration, the configuration dialogs will appear one after the other, guided by a wizard (button **Next**). The settings made in the first dialog will automatically be inherited to the other ones. So those just have to be edited if modificiations are necessary.

Cancel will close the dialog without saving the done modifications in the currently opened dialog (the settings made in the previous dialogs will be kept anyway). You return to the main dialog 'Options' 'Project source control'.

If you have modified an already existing configuration, then the new settings (for all three dialogs) can be saved by pressing **OK**. After that the dialog will be closed and you return to the main dialog 'Options' 'Project source control'.

Macros

If you choose this category , the following dialog box opens:

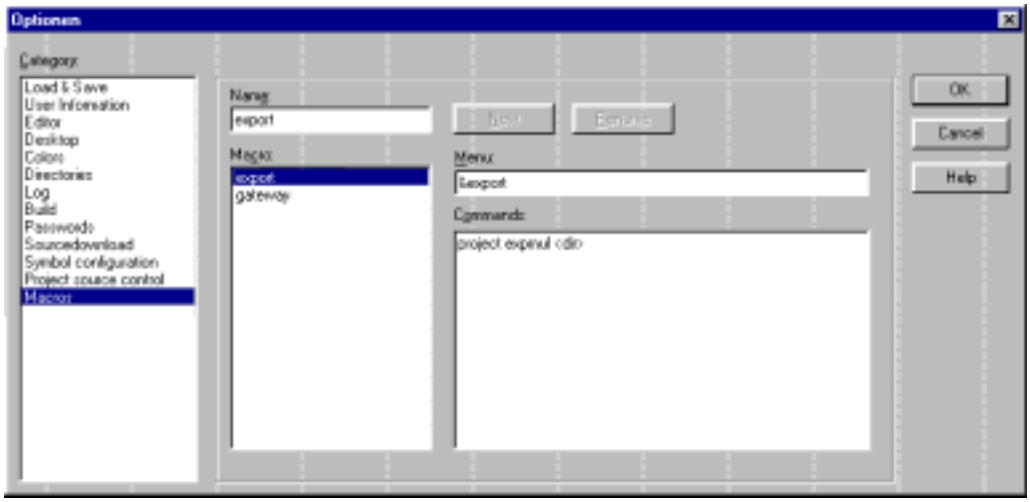


Image 4.22: Options dialog for the category Macros

In this dialog, macros can be defined using the commands of the **CoDeSys** batch mechanism, which can then be called up in the 'Edit' 'Macros' menu (see Chapter 2.5, General Editing functions).

Perform the following steps to define a new macro:

1. In the input field **Name**, you enter a name for the macro to be created. After the **New** button is pressed, this name is transferred into the **Macrolist** field and marked as selected there. The macro list is represented in a tree structure. The locally defined macros are positioned one below the other. If macro libraries (see below) are integrated, then the library names will be listed and by a mouse-click on the plus- resp. minus-signs in front of those entries you can open or close a list of the library elements.
2. The **Menu** field is used to define the menu entry with which the macro will appear in the 'Edit' 'Macros' menu. In order to be able to use a single letter as a short-cut, the letter must be preceded by the symbol '&'. Example: the name "Ma&cro 1" generates the menu entry "Macro 1". Example: the name "Ma&cro 1" will create a menu item "Macro 1".
3. In the editor field **Commands** you define and/or edit the commands that are to constitute the newly created or selected macro. All the commands of the **CoDeSys** batch mechanism and all keywords which are valid for those are allowed. You can obtain a list by pressing the **Help** button. A new command line is started by pressing <Ctrl><Enter>. The context menu with the common text editor functions is obtained by pressing the right mouse button. Command components that belong together can be grouped using quotation marks.
4. If you want to create further macros, perform steps 1-3 again, before you close the dialog by pressing the OK-button.

If you want to delete a macro, select it in the macro list and press button <Del>.

If you want to rename a macro, select it in the macro list, insert a new name in the edit field 'Name' and then press button **Rename**.

To **edit** an existing macro, select it in the macro list and edit the fields 'Menu' and/or 'Commands'. The modifications will be saved when pressing the OK-button.

As soon as the dialog is closed by pressing the **OK**-button the actual description of all macros will be saved in the project.

The macro menu entries in the 'Edit' 'Macros' menu are displayed in the order in which they were defined.

The macros are not tested until a menu selection is made.

Macro libraries:

Macros can be saved in external macro libraries. These libraries can be included in other projects.

- Creating a macro library containing the macros of the currently opened project: Press button **Create**. You get the dialog **Merge project**, where all available macros are listed. Select the desired entries and confirm with OK. The selection dialog will close and dialog **Save Macrolibrary** will open. Insert here a name and path for the new library and press button **Save**. The library will be created named as **<library name>.mac** and the dialog will be closed.
- Including a macro library <library name>.mac in the currently opened project: Press button **Include**. The dialog **Open Macrolibrary** will open, which shows files with extension *.mac. Select the desired library and press button **Open**. The dialog will be closed and the library will be added to the tree of the Macrolist.

Hint: The macros of a project also can be exported ('Project' 'Export').

4.3 Managing Projects

The commands which refer to entire project are found under the menu items 'File' and 'Project'. Some of the commands under 'Project' deal with objects and are therefore described in chapter 2.4, Managing Objects.

'File' 'New'

Symbol:

With this command you create an empty project with the name "Untitled". This name must be changed when saving.

'File' 'Open'

Symbol:

With this command you open an already existing project. If a project has already been opened and changed, then **CoDeSys** asks whether this project should be saved or not.

The dialog box for opening a file appears, and a project file with the extension "*.pro" or a library file with the extension "*.lib" must be chosen. This file must already exist. It is not possible to create a project with the command **"Open"**.

To upload a project file from the PLC, press **PLC** at **Open project from PLC**. You will obtain, as next, the dialog Communication parameters (see menu **‘Online‘ ‘Communication parameters‘**) for setting the transmission parameters when no connection exists yet to the PLC. Once an on-line connection has been created, the system checks whether the same named project files already exist in the directory on your computer hard disc. When this is the case you receive the dialogue **Load the project from the controller** where you can decide whether the local files should be replaced by those being used by the controller. (This sequence is the reverse of the sequence of 'Online' 'Load source code', with which the project source file is stored in the controller. Do not confuse with 'Create Boot project'!)

Note: Please note, that you in any case have to give a new name to a project, when you load it from the PLC to your local directory, otherwise it is unnamed.

If there has not yet been loaded a project to the PLC, you get an error message.

(See also 'Project' 'Options' category 'Sourcedownload').

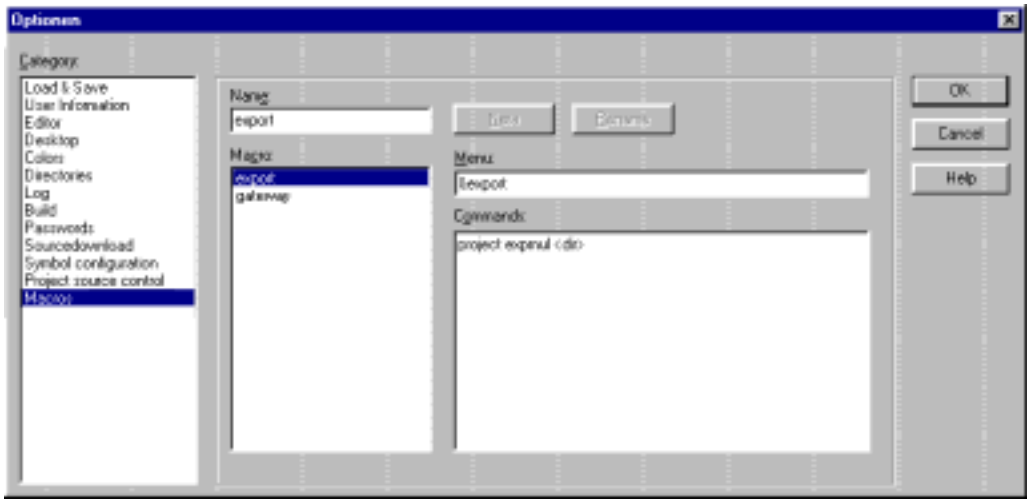


Image 4.23: Standard dialog box for opening a file in CoDeSys

The option **Open project from Source code manager** can be used to open a project which is stored in a ENI project data base. It is a precondition that you have access to an ENI Server which serves the data base. Press button **ENI...** , to get a dialog where you can connect to the server concerning the data base category 'Project objects'.

Insert the appopriate access data (TCP/IP-Address, Port, User name, Password, Read only) and the data base folder (Project name) from which the objects should be get and confirm with **Next**. The dialog will be closed and another one will open where you have to insert the access data for the data base category 'Shared objects'. If you press button **Finish** the dialog will be closed and the objects of the defined folders will automatically be retrieved and displayed in the CoDeSys Object manager. If you want to continue to keep the project objects under data base control, then open the Project options dialogs to set the desired parameters.

The most recently opened files are listed under the command **'File' 'Exit'**. If you choose one of them, then this project is opened.

If Passwords or User groups have been defined for the project, then a dialog box appears for entering the password.

'File' 'Close'

With this command you close the currently-open project. If the project has been changed, then **CoDeSys** asks if these changes are to be saved or not.

If the project to be saved carries the name "Untitled", then a name must be given to it (see '**File' 'Save as**').

'File' 'Save'

Symbol: **Shortcut: <Ctrl>+<S>**

With this command you save any changes in the project. If the project to be saved is called "Untitled", then you must give it a name (see 'File' 'Save as').

'File' 'Save as'

With this command the current project can be saved in another file or as a library. This does not change the original project file.

After the command has been chosen the Save dialog box appears. Choose either an existing **File name** or enter a new file name and choose the desired **file type**.

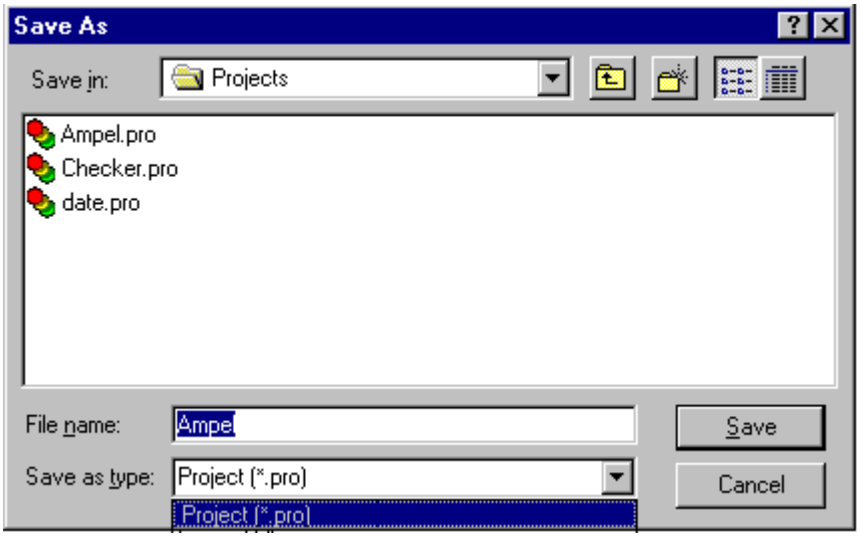


Image 4.24: Dialog box for Save as

If the project is to be saved under a new name, then choose the file type **CoDeSys Project (*.pro)**.

If you choose the file type **Project Version 1.5 (*.pro), 2.0 (*.pro), 2.1 (*.pro) or 2.2 (*.pro)**, then the current project is saved as if it were created with the version 1.5, 2.0, 2.1 or 2.2. Specific data of the version 2.2 can thereby be lost! However, the project can be executed with the version 1.5, 2.0, 2.1 or 2.2.

You can also save the current project as a library in order to use it in other projects. Choose the file type **Internal library (*.lib)** if you have programmed your POUs in **CoDeSys**.

Choose the file type **External library (*.lib)** if you want to implement and integrate POUs in other languages (e.g. C). This means that another file is also saved which

receives the file name of the library, but with the extension "*.h". This file is constructed as a C header file with the declarations of all POUs, data types, and global variables. If external libraries are used, in the simulation mode the implementation, written for the POUs in CoDeSys, will be executed. Working with the real hardware the implementation written in C will be executed.

Licensing a library::

If you want save the project as a licensed library, you can add the appropriate licensing information in the dialog 'Edit Licensing Information'. Open the dialog by pressing the button **Edit license info...**. See for a description in the documentation on the 'License Management in CoDeSys' (see Chapter 8).

After having done all settings, press **OK**. The current project is saved in the indicated file. If the new file name already exists, then you are asked if you want to overwrite this file.

When saving as a library, the entire project is compiled. If an error occurs thereby, then you are told that a correct project is necessary in order to create a library. The project is then not saved as a library.

'File' 'Save/Mail Archive'

This command is used to set up and create a project archive file. All files which are referenced by and used with a **CoDeSys** project can be packed in a compressed zip file. The zip file can be stored or can be directly sent in an email. This is useful if you want to give forward a set of all project relevant files.

When the command is executed, the dialog box '**Save Archive**' opens:

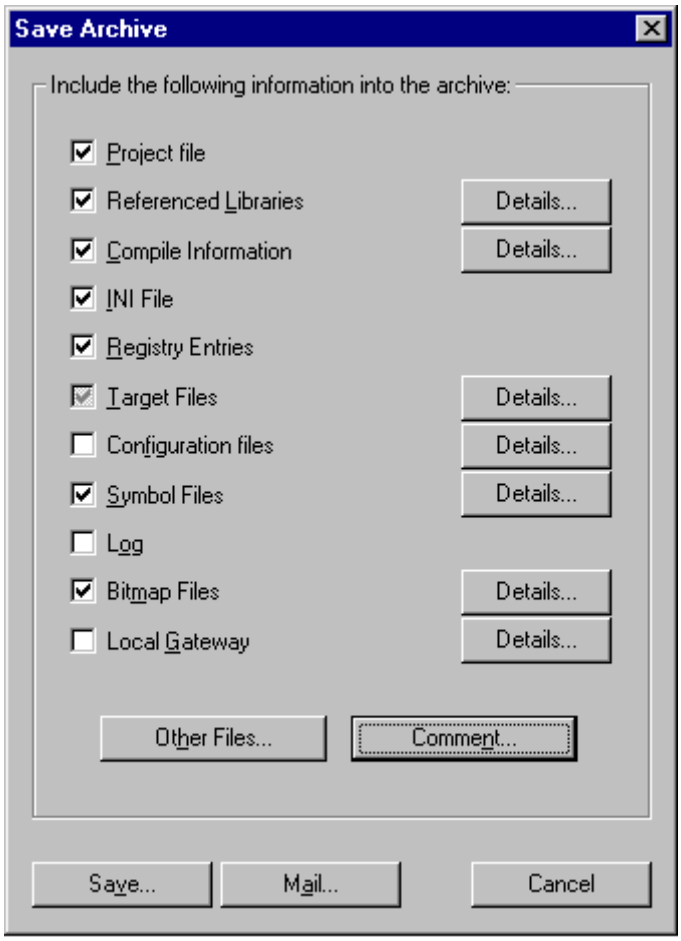


Image 4.25: Dialog box for Setting up an Archive ZIP

Here you can define which file categories should be added to the archive zip file: Select or deselect a category by activating/deactivating the corresponding checkbox. Do this by a single mouseclick in the checkbox or by a doubleclick on the category name. If a category is marked with ☑, all files of this category will be added to the zip file, if it is marked with ☐, none of the files will be added. To select single files of a category press the corresponding button Details.

The dialog '**Details**' will open with a list of available files:

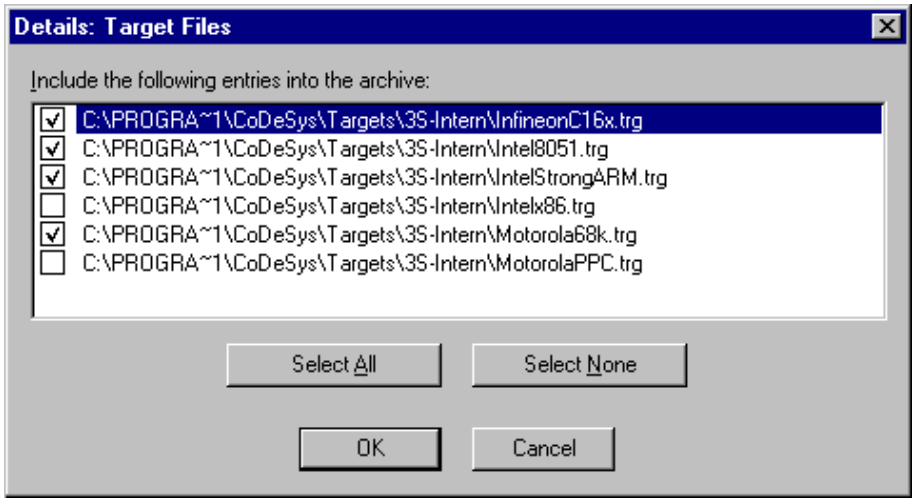


Image 4.26: Dialog box for detailled selection of files for the Archive ZIP

In this dialog select/deselect the desired files: Use the button **Select All** or **Select None** to affect the complete list. A single file can be selected/deselected by a mouseclick in the checkbox, also by a doubleclick on the list entry or by pressing the spacebar when the list entry is marked.

Close the Details dialog with **Save** to store the new settings.

In the main dialog the checkbox of categories, for which not all files are selected, will appear with a grey background color .

The following file categories are available, the right column of the table shows which files can be added to the zip file:

Project File	projectname.pro (the **CoDeSys** project file)
Referenced Libraries	*.lib, *.obj, *.hex (libraries and if available the corresponding object and hex-files)
Symbol Files	*.sdb, *.sym (symbolic information)
Compile Information	*.ci (compile information), *.ri (download/reference information) <temp>.* (temporary compile and download files) also for simulation
Log	*.log (project log file)
INI File	Codesys.ini
Configuration files	files used for PLC configuration (configuration files, device files, icons etc.): e.g. *.cfg, *.con, *.eds, *.dib, *.ico ....
Target Files	*.trg (target files in binary format for all installed targets) *.txt (target files for the installed targets in text format, if available)
Registry Entries	Registry.reg (Entries for Automation Alliance, Gateway und SPS; the following subtrees will be packed: HKEY_LOCAL_MACHINE\SOFTWARE\3S-Smart Software Solutions HKEY_LOCAL_MACHINE\SOFTWARE\AutomationAlliance“
Bitmap Files	*.bmp (bitmaps for project POUs and visualizations)
Gateway Files	Gateway.exe, GatewayDDE.exe, GClient.dll, GDrvBase.dll, GDrvStd.dll, GHandle.dll, GSymbol.dll, GUtil.dll, further DLLs in the gateway directory if available

To add any other files to the zip, press the button **Other Files**. The dialog 'Other files' will open where you can set up a list of desired files.

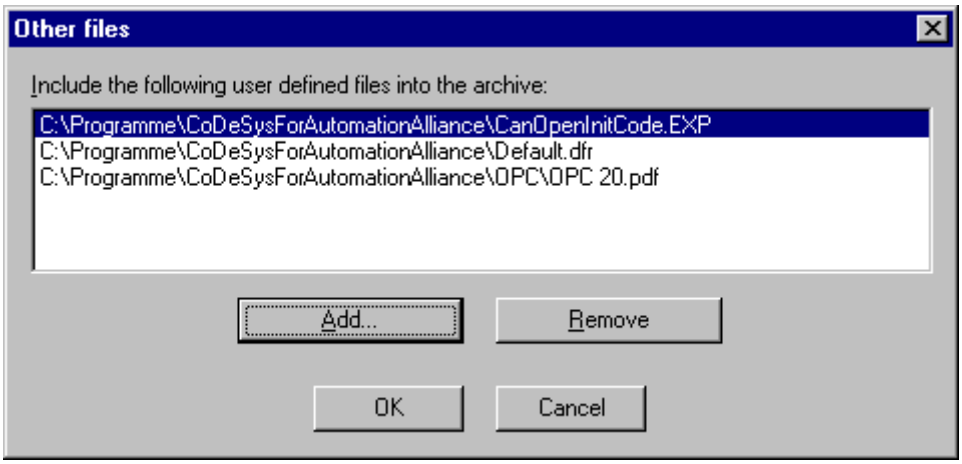


Image 4.27: Dialog box for adding other files for the Archive ZIP

Press the button Add to open the standard dialog for opening a file, where you can browse for a file. Choose one and confirm with Open. The file will be added to the list in the 'Other files' dialog. Repeat this for each file you want to add. To delete entries from the list, press the button Remove. When the list of selected files is ok, close the dialog with OK.

To add a Readme file to the archive zip, press the button **Comment**. A text editor will open, where you can enter any text. If you close the dialog with OK, during creation of the zip file a **readme.txt** file will be added. Additionally to the entered comments it will contain information about the build date and version of **CoDeSys**.

If all desired selections have been made, in the main dialog press

- **Save...** to create and save the archive zip file: The standard dialog for saving a file will open and you can enter the path, where the zip should be stored. The zip file per default is named **<projectname>.zip**. Confirm with **Save** to start building it. During creation the current progress status is displayed and the subsequent steps are listed in the message window.
- **Mail…** to create a temporary archive zip and to automatically generate an empty email which contains the zip as an attachment. This feature only works if the MAPI (Messaging Application Programming Interface) has been installed correctly on the system, otherwise an error message is generated. During setup of the email the progressing status is displayed and the steps of the action are listed in the message window. The temporary zip file will be removed automatically after the action has been finished.
- **Cancel** to cancel the action; no zip file will be generated.

'File' 'Print'

Shortcut: <Ctrl>+<P>

With this command the content of the active window is printed.

After the command has been chosen, then the Print dialog box appears. Choose the desired option or configure the printer and then click **OK**. The active window is printed. Color output is available from all editors.

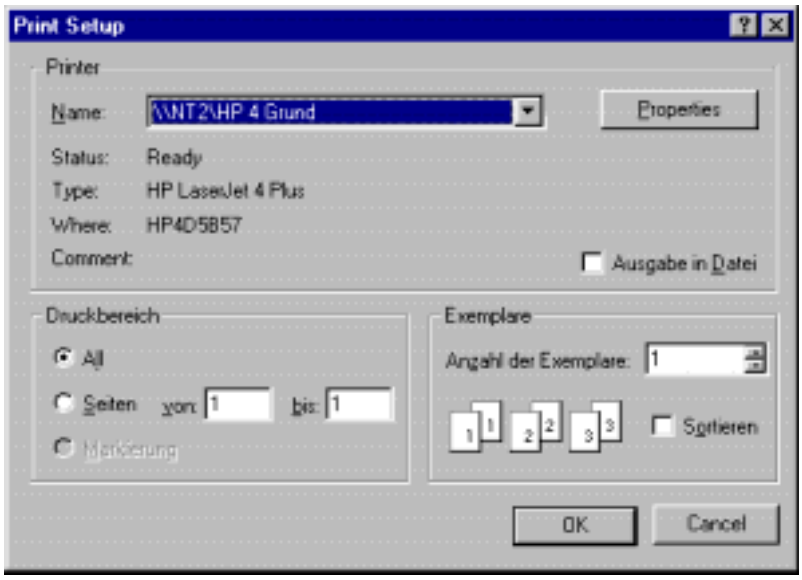


Image 4.28: Print dialog box

You can determine the **number of the copies** and print the version to a file.

With the button **Properties** you open the dialog box to set up the printer.

You can determine the layout of your printout with the command 'File' 'Printer Setup'.

During printing the dialog box shows you the number of pages already printed. When you close this dialog box, then the printing stops after the next page.

In order to document your entire project, use the command **'Project' 'Document'**.

If you want to create a for your project, in which you can store comments regarding all the variables used in the project, then open a global variables list and use the command **'Extras' 'Make docuframe file'** (see Chapter 4.1.3, Document Frame).

'File' 'Printer setup'

With this command you can determine the layout of the printed pages. The following dialog box is now opened:

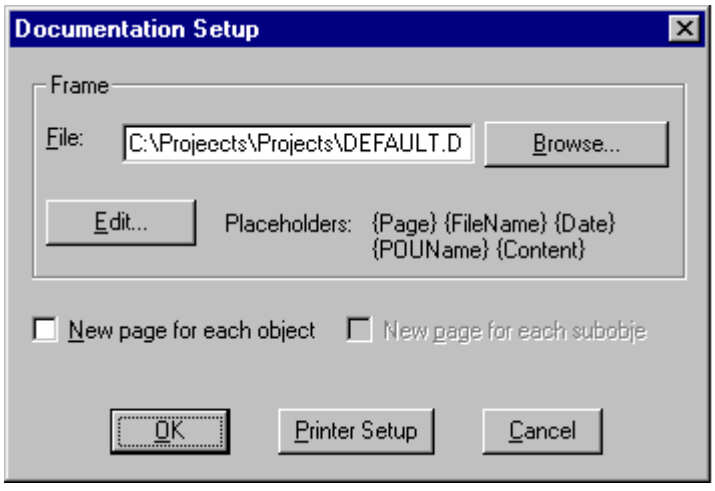


Image 4.29: Page Layout Dialog Box

In the field **File** you can enter the name of the file with the extension ".dfr" in which the page layout should be saved. The default destination for the settings is the file DEFAULT.DFR. If you would like to change an existing layout, then browse through the directory tree to find the desired file with the button **Browse.**

You can also choose whether to begin a **new page for each object** and **for each subobject**. Use the **Printer Setup** button to open the printer configuration.

If you click on the **Edit** button, then the frame for setting up the page layout appears. Here you can determine the page numbers, date, filename and POU name, and also place graphics on the page and the text area in which the documentation should be printed.

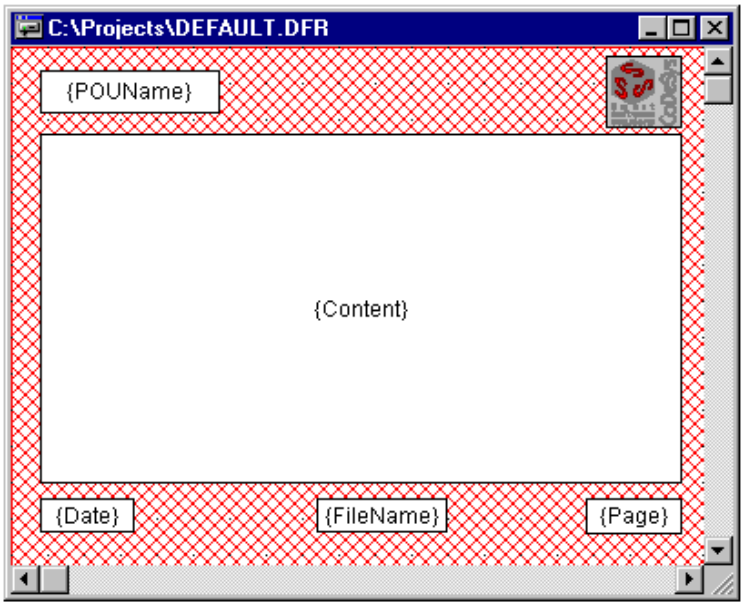


Image 4.30: Window for pasting the placeholders on the page layout

With the menu item **'Insert' 'Placeholder'** and subsequent selection among the five placeholders (**Page, POU name, File name, Date, and Content**), insert into the layout a so-called placeholder by dragging a rectangle[2] on the layout while pressing the left mouse button. In the printout they are replaced as follows:

Command	Placeholder	Effect
Page	{Page}	Here the current page number appears in the printout.
POU name	{POU Name}	Here the current name of the POU appears.
File name	{File Name}	Here the name of the project appears.
Date	{Date}	Here the current date appears.
Contents	{Contents}	Here the contents of the POU appear.

In addition, with **'Insert' 'Bitmap'** you can insert a bitmap graphic (e.g. a company logo) in the page. After selecting the graphic, a rectangle should also be drawn here

[2] Drawing a rectangle on the layout by dragging the mouse diagonally while pressing the left mouse button.

on the layout using the mouse. Other visualization elements can be inserted (see Chapter 5, Visualization).

If the template was changed, then **CoDeSys** asks when the window is closed if these changes should be saved or not.

'File' 'Exit'

Shortcut: <Alt>+<F4>

With this command you exit from **CoDeSys**.

If a project is opened, then it is closed as described in **'File' 'Save'**.

'Project' 'Build'

Shortcut: <F11>

The project is compiled using 'Project' 'Build'. The compilation process is basically incremental, that is only changed POUs are recompiled. A non-incremental compilation can also be obtained if the command 'Project' 'Clear all' is first executed.

For target systems that support Online Change, all POUs that will be loaded into the controller on the next download are marked with a blue arrow in the Object Organizer after compilation.

The compilation process that is carried out with 'Project' 'Build' occurs automatically if the controller is logged-in via 'Online' 'Log-in'.

During compilation a message window is opened which shows the progress of the compilation process and any errors and warnings which may occur during compilation. Errors and warnings are marked with numbers. Using F1 you get more information about the currently selected error.

See Appendix Appendix K: for a listing of all available error messages and warnings.

Data allocation
Task Configuration
Implementation of POU 'PLC_PRG'
Warning 1302: New externally referenced functions inserted. Online Change is therefore no longer possible!
Hardware-Configuration
0 Error(s), 1 Warning(s).

Image 4.31: Message window of a project

If the option **Save before compilation** is selected in the options dialog of the Load & Save category, the project is stored before compilation.

Note: Cross references are created during compilation and are stored with the compilation information. In order to be able to use the commands **'Show Call Tree'**, **'Show Cross Reference'** and the commands **'Unused Variables'**, **'Concurrent Access'**, and **'Multiple Write Access on output'** in the **'Project' 'Check'** menu, the project must be rebuilt after any change.

‘Project’ ’Rebuild all’

With 'Project' 'Rebuild all', unlike the incremental compilation ('Project' 'Build'), the project is completely recompiled. Downoad-Information is not discarded, however, as is the case with the command 'Clear All'.

‘Project’ 'Clean all’

With this command, all the information from the last download (Download-Information) and from the last compilation is deleted.

After the command is selected a dialog box appears, reporting that Online Change is no longer possible. At this point the command can either be cancelled or confirmed.

Note: After 'Clear all' a login on the PLC project is only possible if the *.ri file with the project information from the last download was first explicitly saved outside the project directory (see 'Load Download-Information') and can now be reloaded prior to logging-in.

'Project' 'Load Download-Information'

With this command the Download-Information belonging to the project can get reloaded, if it was saved to a directory different from that where the project is. After choosing the command the standard dialogue 'File Open' opens.

The Download-Information is saved automatically at each download to a file, which is named <project name><target identifier>.ri and which is put to the project directory. This file is loaded, when the project is opened and at login it is used to check whether the PLC project is fitting to the currently opened CoDeSys project (Id-check). Furthermore it is used to check, in which POUs the code has been changed. In systems which support the online change functionality, then only these POUs will be loaded to the PLC during online change procedure.

But: If the *.ri-file in the project directory gets deleted by the command '**Project**' '**Clean all**', you only can reload the Download-Information, if you had stored the *.ri-file in another directory too.

‘Project’ 'Translate into another language‘

This menu item is used for translating the current project file into another language. This is carried out by reading in a translation file that was generated from the project and externally enhanced in the desired national language with the help of a text editor.

Two menu sub-items are present:

- Create translation file
- Translate project

Create translation file

This command in the 'Project' 'Translate into another language' menu leads to the 'Create translation file' dialog.

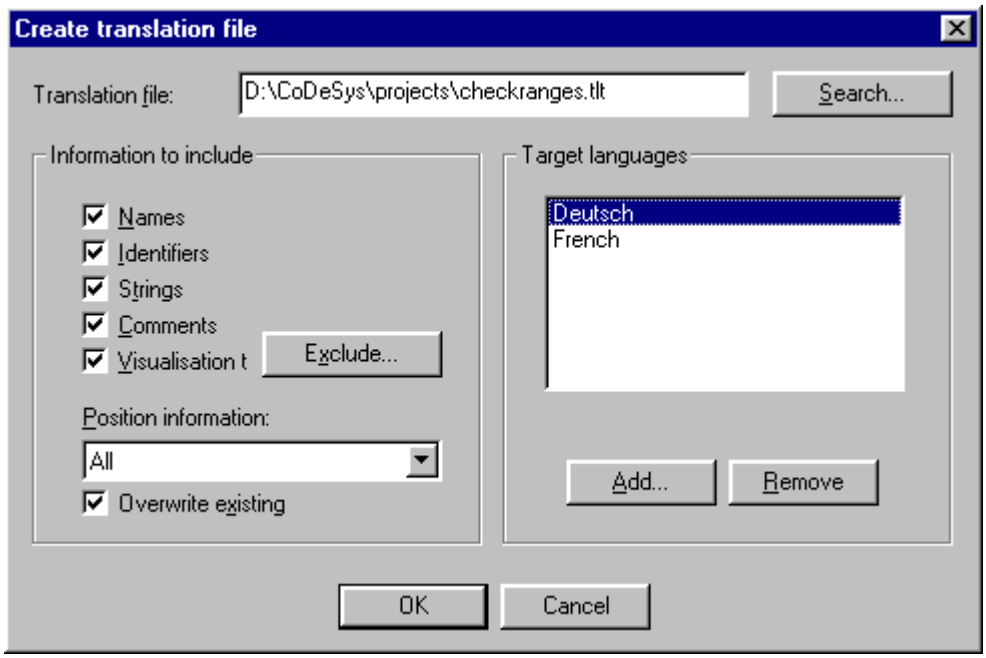


Image 4.32: Dialog for creating a translation file

In the **Translation file** field, enter a path that shows where the file is to be stored. The default file extension is *.tlt; this is a text file.

If there already exists a translation file which you want to process, give the path of this file or use the **Search** button to reach the standard Windows file selection dialog.

The following information from the project can optionally be passed to the translation file that is being modified or created, so that they will be available for translation: **Names** (names, e.g. the title 'POUs' in Object Organizer), **Identifiers, Strings, Comments, Visualisation texts.** In addition, **Position information** for these project elements can be transferred.

If the corresponding options are checked, the information from the current project will be exported as language symbols into a newly created translation file or added to an already existing one. If the respective option is not selected, information belonging to the pertinent category, regardless of which project it came from, will be deleted from the translation file.

The “Text” and “Tooltip-Text” elements in the visualization elements are considered here to be visualization texts.

Note: For visualization texts (‚Text' and ‚Text for Tooltip’ in the visualization elements) it must be noted that they must be bracketed by two “#” symbols in the configuration dialog of the visualization element (e.g. #text#) in order to be transferred to the translation file. (See in this connection chapter 8, Visualization). These texts are also not translated with the command 'Project' 'Translate into other languages' ! A language change for the visualization can only occur in Online mode if the corresponding language is entered in the 'Extras' 'Settings' dialog.

Position information: This describes with the specifications file path, POU and line the position of the language symbol made available for translation. Three options are available for selection:

None:	No position information is generated.
First appearance:	The position on which the element first appears is added to the translation file.
All:	All positions on which the corresponding element appears are specified.

If a translation file created earlier is to be edited which already contains more position information than that currently selected, it will be correspondingly truncated or deleted, regardless of which project it was generated from.

Note: A maximum of 64 position specifications will be generated per element (language symbol), even if the user has selected “All” under “Position Information” in the ‚Create Translation File' dialog.

Overwrite existing: Existing position information in the translation file, that is currently being processed, will be overwritten, regardless of which project generated it.

Target languages: This list contains identifiers for all languages which are contained in the translation file, as well as those to be added upon completion of the 'Create translation file' dialog.

The **Exclude** button opens the 'Exclude libraries' dialog. Here, libraries included to the project can be selected, whose identifier information is not to be transferred to the translation file. To accomplish this, the corresponding entry in the table **Included libraries** on the left is selected with the mouse and placed in the **Excluded libraries** table to the right using the **Add** button. Likewise, entries already placed there can be removed using the **Remove** button. OK confirms the setting and closes the dialog.

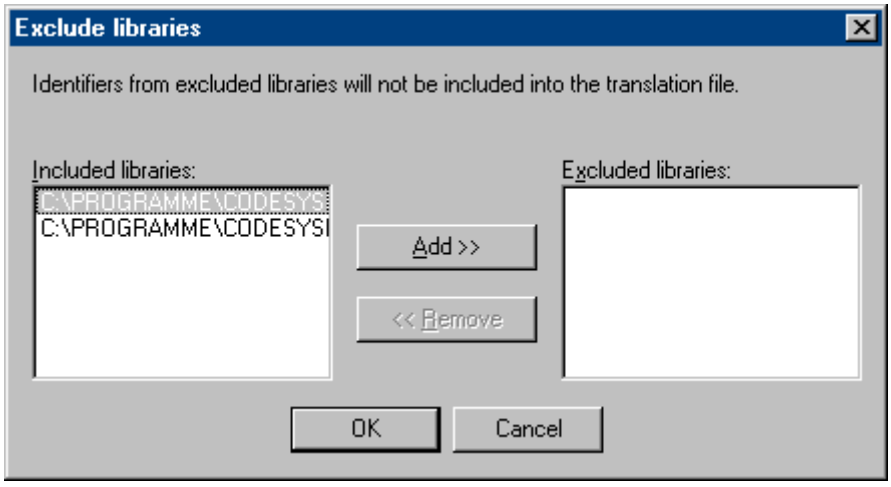


Image 4.33: Dialog for excluding library information for the translation file

The **Add** button opens the **'Add Target Language'** dialog:

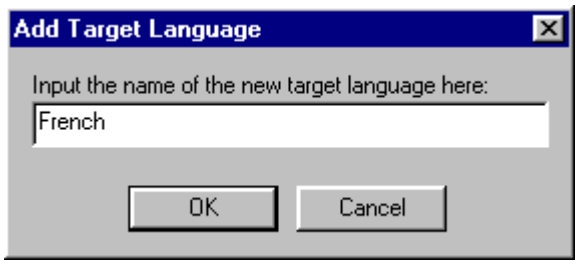


Image 4.34: Dialog for adding a target language (Project, Translate into Another Language)

A language identifier must be entered into the editor field; it may not have a space or an umlaut character (ä, ö, ü) at either the beginning or the end.

OK closes the 'Add Target Language' dialog and the new target language appears in the target language list.

The **Remove** button removes a selected entry from the list.

You may also confirm the “Create translation file” dialog via **OK**, in order to generate a translation file.

If a translation file of the same name already exists you will get the following confirmation message to be answered Yes or No:

" The specified translation file already exists. It will now be altered and a backup copy of the existing file will be created. Do you want to continue?"

No returns you without action to the 'Create translation file' dialog. If **Yes** is selected, a copy of the existing translation file with the filename “Backup_of_<translation file>.xlt” will be created in the same directory and the corresponding translation file will be modified in accordance with the options that have been entered.

The following takes place when a translation file is generated:

For each new target language, a placeholder (”##TODO”) is generated for each language symbol to be displayed.

If an existing translation file is processed, file entries of languages that appear in the translation file, but not in the target language list, are deleted, regardless of the project from which they were generated.

Editing of the translation file

The translation file must be opened and saved as a text file. The signs ## mark keywords. The ##TODO-placeholders in the file can be replaced by the valid translation. For each language symbol a paragraph is generated which starts with a ##NAME_ITEM and ends with a ##END_NAME_ITEM. (For comments correspondingly ##COMMENT_ITEM etc.).

See in the following an example of a translation file paragraph which handles the name of one of the POUs of the project. ST_Visu. The source language is German, the target languages shall be English(USA) and French. In this examle the position information of the project element which should be translated has been added:

before translation:

```
##NAME_ITEM
[D:\CoDeSys\projects\Bspdt_22.pro::ST_Visualisierung::0]
ST_Visualisierung
##English :: ##TODO
```

```
##French :: ##TODO
##END_NAME_ITEM
```

after translation:

The ##TODOs have been replaced by the English resp. French word for 'Visualisierung':

```
##NAME_ITEM
[D:\CoDeSys\projects\Bspdt_22.pro::ST_Visualisierung::0]
ST_Visualisierung
##English :: ST_Visualization
##French :: ST_Visu
##END_NAME_ITEM
```

Please check that the translated Identifier and Names remain valid concerning the standard and that strings and comments are in correct brackets. Example: For a comment (##COMMENT_ITEM) which is represented with "(* Kommentar 1)" in the translation file, the "##TODO" behind "##English" must be replaced by a "(* comment 1 *)". For a string (##STRING_ITEM) represented with "zeichenfolge1" the "##TODO" must be replaced by "string1".

Hint: The following parts of a translation file should not be modified without detailed knowledge: Language block, Flag block, Position information, Original texts.

Translate Project (into another Language)

This command in the 'Project' 'Translate into Another Language' menu opens the 'Translate Project into Another Language' dialog.

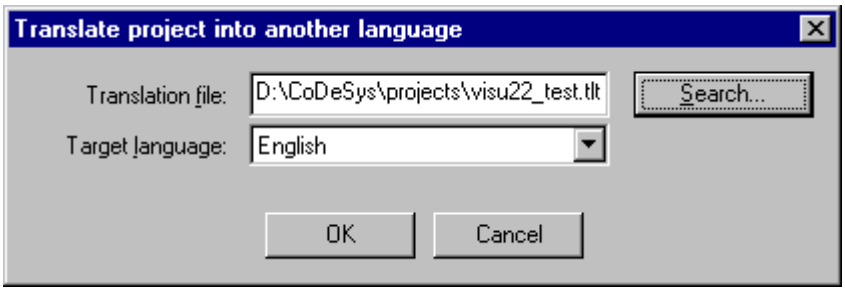


Image 4.35: Dialog for translating the project into another language

The current project can be translated into another language if an appropriate translation file is used.

Note: If you want to save the version of the project in the language in which it was originally created, save a copy of the project prior to translation under a different name. The translation process cannot be undone.

In the field **Translation file**, provide the path to the translation file to be used. By pressing **Search** you may access the standard Windows file selection dialog.

The field **Target language** contains a list of the language identifiers entered in the translation file, from which you can select the desired target language.

OK starts the translation of the current project into the chosen target language with the help of the specified translation file. During translation, a progress dialog is displayed, as well as error messages, if any. After translation, the dialog box and all open editor windows of the project are closed.

Cancel closes the dialog box without modification to the current project.

If the translation file contains erroneous entries, an error message is displayed after OK is pressed, giving the file path and the erroneous line, e.g.: “[C:\Programs**CoDeSys**\projects\visu.tlt (78)]; Translation text expected”

'Project' 'Document'

This command lets you print the documentation of your entire project. The elements of a complete documentation are:

- The POUs,
- the contents of the documentation,
- the data types,
- the visualizations
- the resources ,global variables, variables configuration, the Sampling Trace, the PLC Configuration, the Task Configuration, the Watch and Receipt Manager)
- the call trees of POUs and data types, as well as
- the cross reference list.

For the last two items the project must have been built without errors.

Only those areas in the dialog box are printed which are highlighted in blue.

If you want to select the entire project, then select the name of your project in the first line.

If, on the other hand, you only want to select a single object, then click on the corresponding object or move the dotted rectangle onto the desired object with the arrow key. Objects which have a plus sign in front of their symbols are organization objects which contain other objects. With a click on a plus sign organization object is expanded, and with a click on the resulting minus sign it can be closed up again. When you select an organization object, then all relevant objets are also selected. By pressing the <Shift> key you can select a group of objects, and by pressing the <Ctrl> key you can select several individual objects.

Once you have made your selection, then click on **OK**. The Print dialog box appears. You can determine the layout of the pages to be printed with **'File' 'Printer setup'**.

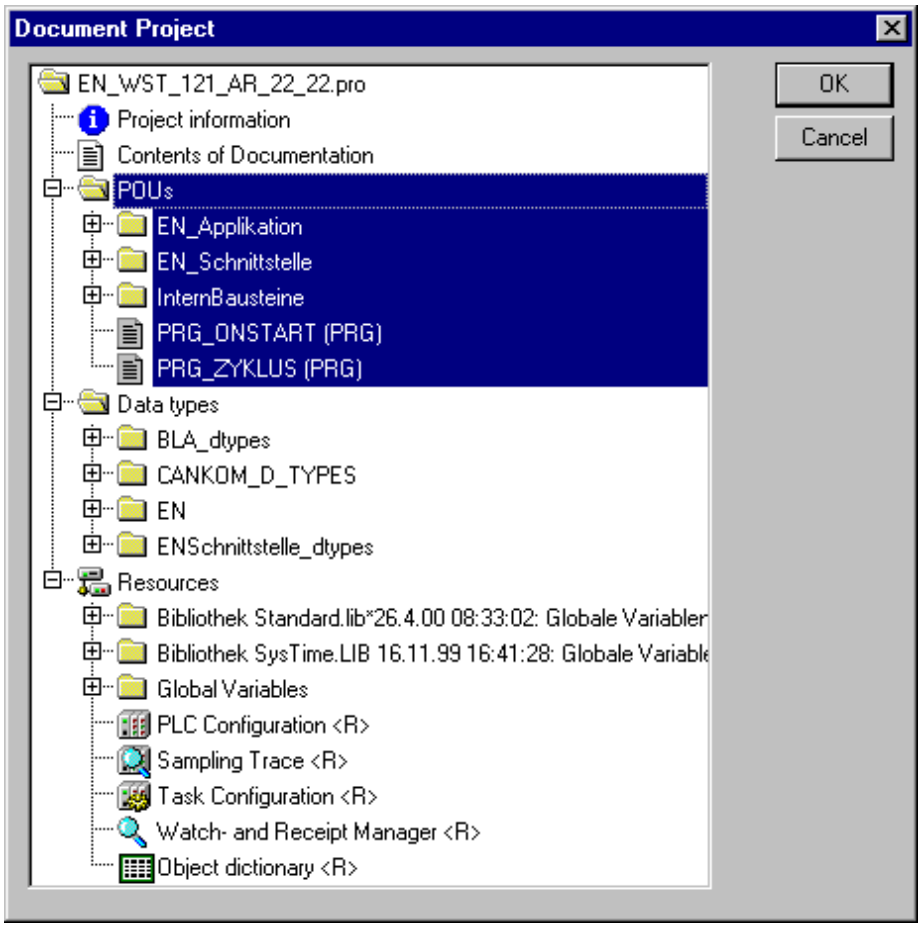


Image 4.36: Dialog box for project documentation

'Project' 'Export'

With **CoDeSys** projects can be exported or imported. That allows you to exchange programs between different IEC programming systems.

There is a standardized exchange format for POUs in IL, ST, and SFC (the Common Elements format of IEC 1131-3). For the POUs in LD and FBD and the other objects **CoDeSys** has its own filing format since there is no text format for this in IEC 1131-3.

The selected objects are written to an ASCII file.

POUs, data types, visualizations, and the resources can be exported. In addition, entries in the library manager, that is the linking information to the libraries, can be exported (not the libraries themselves!).

Important: Re-importing an exported FBD or LD POU results in an error if a comment in the graphical editor contains a single quotation mark ('), as this will be interpreted as the beginning of a string !

Once you have made your selection in the dialog box window (the same way as with **'Project' 'Document'**), you can decide, whether you want to export the selected parts to one file or to export in separate files, one for each object. Switch on or off the option **One file for each object** then click on **OK**. The dialog box for saving files appears. Enter a file name with the expansion ".exp" respectively a directory for

the object export files, which then will be saved there with the file name <objectname.exp>.

'Project' 'Import'

In the resulting dialog box for opening files select the desired export file.

The data is imported into the current project. If an object with the same name already exists in the same project, then a dialog box appears with the question "Do you want to replace it?": If you answer **Yes**, then the object in the project is replaced by the object from the import file. If you answer **No**, then the name of the new objects receives as a supplement an underline and a digit ("_0", "_1", ..). With **Yes, all** or **No, all** this is carried out for all objects.

If the information is imported to link with a library, the library will be loaded and appended to the end of the list in the library manager. If the library was already loaded into the project, it will not be reloaded. If, however, the export file that is being imported shows a different storage time for the library, the library name is marked with a “*” in the library manager (e.g. standard.lib*30.3.99 11:30:14), similar to the loading of a project. If the library can not be found, then an information dialog appears: “Cannot find library {<path>\}<name> <date> <time>”, as when a project is loaded.

In the message window the import is registered.

'Project' 'Siemens Import'

In the submenu "Siemens Import" you find the commands for importing POUs and variables from Siemens-STEP5 and STEP7 files.

The following commands are available:

- "Import from SEQ symbol file
- "Import from S5 file

You will find more detailed information about Siemens import in Appendix H: .

'Import from SEQ symbol file'

The command in the submenu **'Project' 'Siemens Import'** serves to import global variables from STEP5 and STEP7 symbol files. This command should be entered before the command '**Import from S5 file'** so that when importing from POUs symbolic names can be created out of absolute addresses.

In the resulting dialog box for opening files choose the desired symbol file with the extension ".seq".

The data is imported into the current project.

You can find more detailed information about Siemens import in Appendix H: .

'Import from S5 file'

The command in the submenu **'Project' 'Siemens Import'** serves to import POUs out of Siemens S5 program files. The importation of symbolic names occurs, on the other hand, through the SEQ file.

In the resulting dialog box for opening files choose the desired S5 file with the extension ".s5d".

In a subsequent dialog box you can choose from a list of the POUs. You can choose whether the POUs should be left in the STEP5-IL language or directly converted into IL, LD, or FBD.

The POUs are imported into the current project.

You can find more detailed information about Siemens import in Appendix H: .

‘Project‘ ‘Compare‘

This command is used to compare two projects or to compare the actual version of one project with that which was saved last.

Overview:

actual project:	Project, which you are currently working on.
reference project:	Project, which should be compared with the actual project.
compare mode:	in this mode the project will be displayed after the command 'Project' 'Compare' has been executed.
unit:	Smallest unit which can be compared. Can be a line (declaration editor, ST editor, IL editor), a network (FBD editor, LD editor) or a element/POU (CFC ediotr, SFC editor).

After the command has been executed, the actual project and the reference project will be presented in 'compare mode' in a bipartited window. The names of the POUs, for which differences have been found, are marked by color. For editor POUs also the content of the POUs is displayed in a vis-a-vis way. The results and the way of presenting in compare mode depend on: 1. what filters have been activated for the compare run, affecting the consideration of whitespaces and comments during comparison; 2. whether modification within lines or networks or elements are evaluated as a completely new inserting of a POU or not.

The version of the reference project can be accepted for single diffences or for 'all equally marked' differences. To accept means that the version of the reference project is taken over to the actual project.

Please note: In compare mode (see status bar: COMPARE) the project cannot get edited !

Execute comparison:

After executing the command 'Project' 'Compare' the dialog 'Project Comparison' opens:

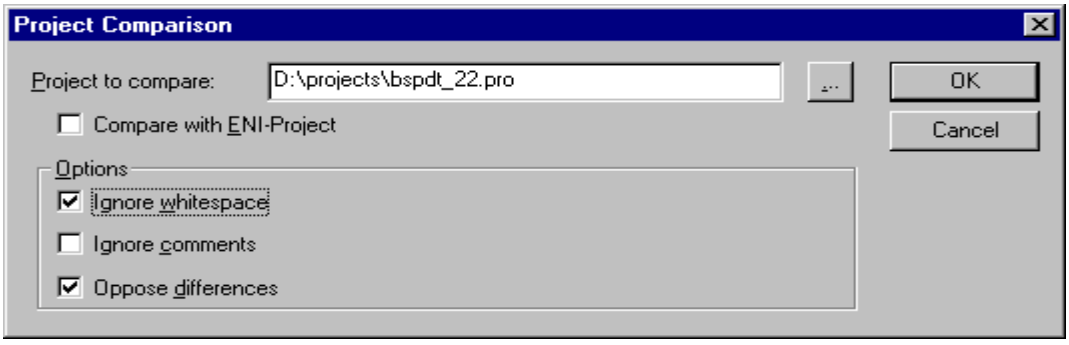


Image 4.37: Dialog for Project Comparison

Insert the path of the reference project at **Project to compare**. Press button if you want to use the standard dialog for opening a project. If you insert the name of the actual project, the current version of the project will be compared with the version which was saved last.

If the project is under source control in an ENI data base (see Chapter 7), then the local version can be compared with the actual version found in the data base. For this activate option **Compare with ENI-Project**.

The following options concerning the comparison can be activated:

Ignore whitespaces: There will be detected no differences which consist in a different number of whitespaces.

Ignore comments: There will be detected no differences in comments.

Oppose differences: If a line, a network or an element wihthin a POU has been modified, in compare mode it will be displayed in the bipartited window directly opposite to the version of the other project (marked red, see below). If the option is deactivated, the corresponding line will be displayed in the reference project as 'deleted' and in the actual project as 'inserted' (blue/green, see below). This means it will not be displayed directly opposite to the same line in the other project.

Example:

Line 0005 has been modified in actual project (left side).

When the dialog Project Comparison is closed by pressing OK, the comparison will be executed according to the settings.

Representation of the result of the comparison:

First the structure tree of the project, titled with "Project Comparison", will be opened to display the results of the comparison. Here you can select particular POUs to see the found differences in detail.

Project overview in compare mode:

After the project compare has been executed, a bipartited window opens which shows the project in compare mode. In the title bar you find the project paths: "Project comparison <path of actual project> - <path of reference project>". The actual project is represented In the left half of the window, the reference project in the right one. Each structure tree shows the projects' name at the uppermost position, apart from that it corresponds to the the object organizer structure.

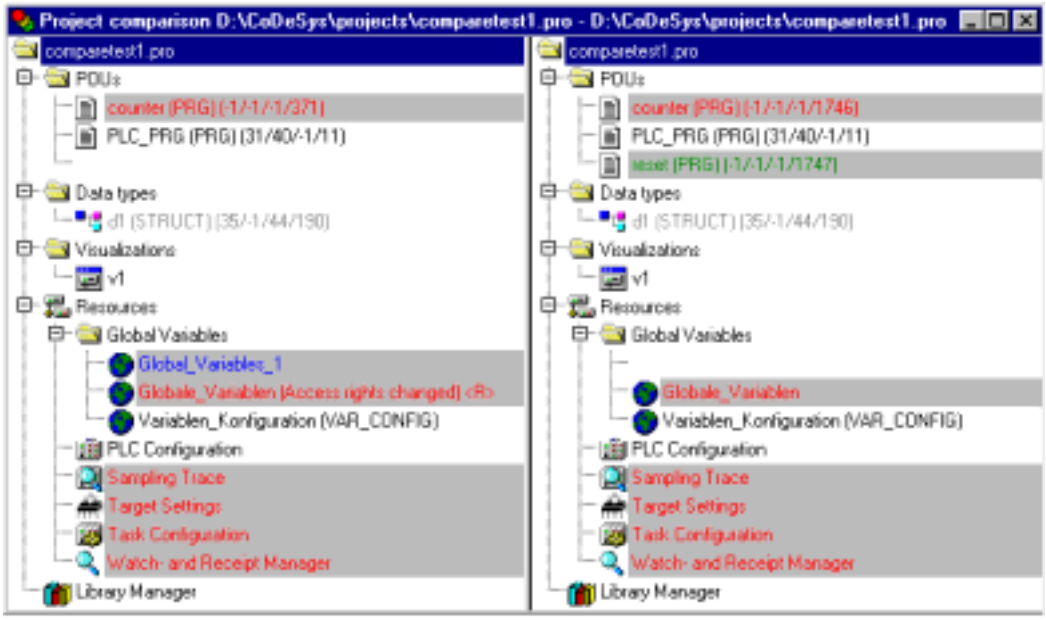


Image 4.38: Example, Project in Compare mode

POUs which are different, are marked in the structure tree by a shadow, a specific color and eventually by an additional text :

Red: Unit has been modified; is displayed with red colored letters in both partitions of the window.

Blue: Unit only available in compare project; a gap will be inserted at the corresponding place in the structure overview of the actual project.

Grün: Unit only available in actual project; a gap will be inserted at the corresponding place in the structure overview of the actual project.

Black: Unit for which no differences have been detected.

"(Properties changed)": This text is attached to the POU name in the project structure tree, if differences in the properties of the POU have been detected.

"(Access rights changed)": This text is attached to the POU name in the project structure tree, if differences in the access rights of the POU have been detected.

2. POU contents in compare mode:

By a double click on a line in the structure overview, which is marked red because of a modification, the POU is opened.

If it is a text or graphic editor POU, it will be opened in a bipartited window. The content of the reference project (right side) is set opposite to that of the actual project (left side). The smallest unit which will be regarded during comparison, is a line (declaration editor, ST, IL), a network (FBD, LD) or an element (CFC, SFC). The same coloring will be used as described above for the project overview. Example:

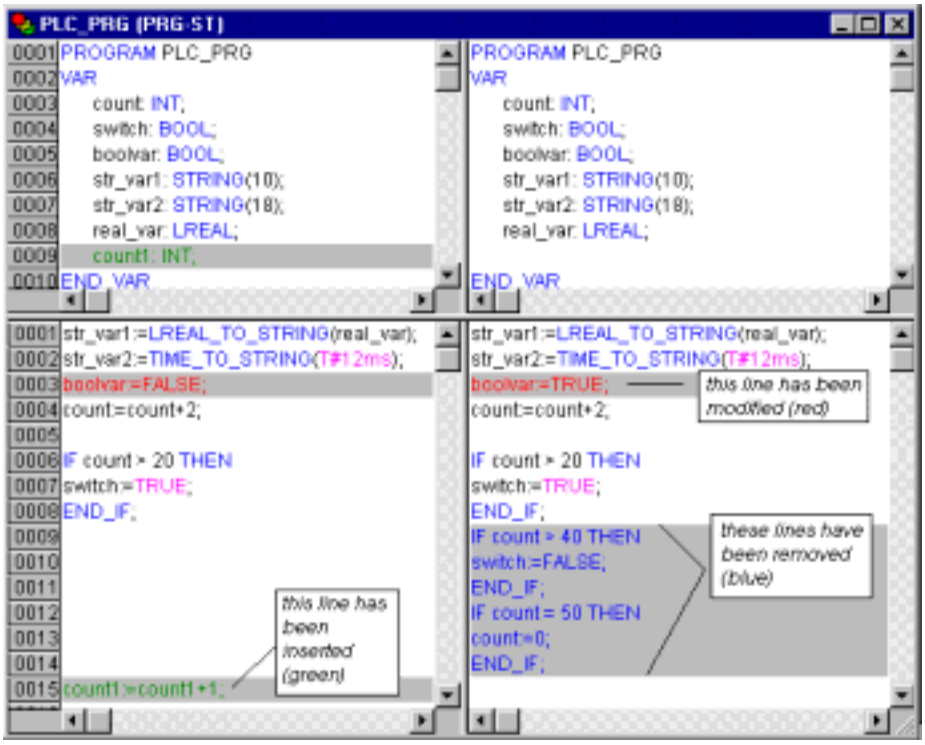


Image 4.39: Example, POU in compare mode

If it is not a editor POU, but the task configuration, the target settings etc., then the POU version of the actual and the reference project can be opened in separate windows by a double click on the respective line in the project structure. For those project POUs no further details of differences will be displayed.

Working in the compare mode (Menu 'Extras', Context menu):

If the cursor is placed on a line in the bipartited window, which indicates a difference, the menu 'Extras' resp. The context menu offers a selection of the following commands, depending on whether working in the project overview or in a POU.

'**Next difference**':	The cursor jumps to the next unit, where a difference is indicated. (line in project overview, line/network/element in POU)
'**Previous difference**':	The cursor jumps to the previous unit, where a difference is indicated . (line in project overview, line/network/element in POU)
'**Accept change**':	For all coherent (e.g. subsequent lines) units, which have the same sort of difference marking, the version of the reference project will be accepted for the actual project. The corresponding units will be shown (with

	the corresponding coloring) in the left side of the window. If it is an unit, which is marked red (just modification), then the acceptance will be recognizable by yellow coloring of the text in the actual project.
'**Accept changed item**':	Only the single unit (line, network, element) where the cursor is currently placed, will be accepted for the actual version. The corresponding units will be shown (with the corresponding coloring) in the left side of the window. If it is an unit, which is marked red (just modification), then the acceptance will be recognizable by yellow coloring of the text in the actual project.
'**Accept properties**':	The object properties for the POU, where the cursor is currently placed, will be accepted for the actual project as they are set in the reference version.
'**Accept access rights**' (only in project overview):	The object access rights for the POU, where the cursor is currently placed, will be accepted for the actual project as they are set in the reference version .

'Project' 'Merge'

With this command you can merge objects (POUs, data types, visualizations, and resources) as well as links to libraries from other projects into your project.

When the command has been given, first the standard dialog box for opening files appears. When you have chosen a file there, a dialog box appears in which you can choose the desired object. The selection takes place as described with **'Project' 'Document'** .

If an object with the same name already exists in the project, then the name of the new object receives the addition of an underline and a digit ("_1", "_2" ...).

'Project' 'Project info'

Under this menu item the information about your project can be saved. When the command has been given, then the dialog box shown in the image below opens.

The following project information is displayed:

- **File** name
- Directory path
- The time of the most recent change (**Change date**)

This information can not be changed.

In addition, you can can add the following information:

- A **Title** of the project,
- the name of the **Author**,
- the **Version** number, and
- a **Description** of the project.

This information is optional.

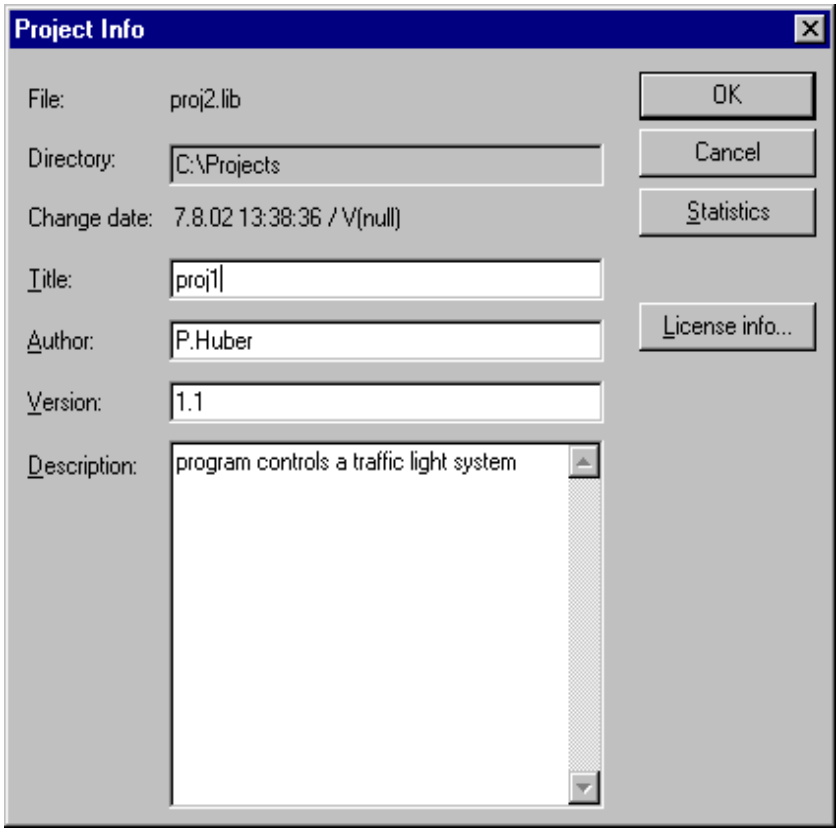


Image 4.40: Dialog box for entering project information

When you press the button **Statistics** you receive statistical information about the project.

It contains information such as the number of the POUs, data types, and the local and global variables as they were traced at the last compilation.

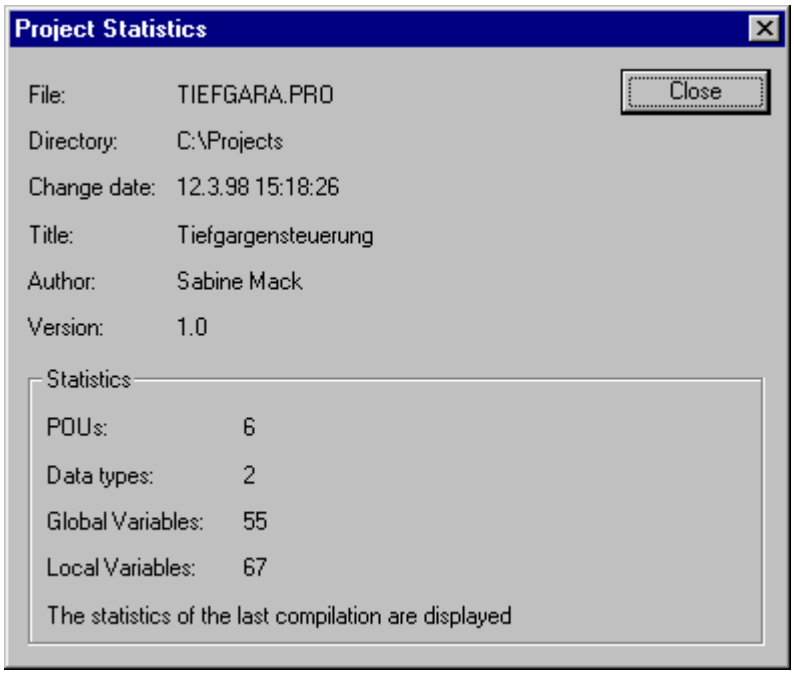


Image 4.41: Example of project statistics

The button **License info** will be available, if you work on a CoDeSys project, which had been saved already with licensing information by the command 'File' 'Save as...'. In this case the button opens the dialog **Edit Licensing Information**, where you can modify or remove the license (see 'License Management in CoDeSys', Chapter 8).

If you choose the option **Ask for project info** in the category **Load & Save** in the Options dialog box, then while saving a new project, or while saving a project under a new name, the project info is called automatically.

'Project' 'Global Search'

With this command you can search for the location of a text in POUs, data types, or in the objects of the global variables.

When the command is entered, a dialog box opens in which you can choose the desired object. The selection is made as in the **'Project' 'Document'** description.

If the selection is confirmed with **OK**, the standard dialog for Search will be opened. This appears immediately when the command 'Global Search' is invoked via the symbol in the menu bar; the search is then automatically carried out in all searchable parts of the project. The most recently entered search strings can be selected through the combo box of the **Search for** field. If a text string is found in an object, the object is loaded into the corresponding editor or in the library manager and the location where the string was found is displayed. The display of the text that is found, as well as the search and find next functions behave similarly to the command 'Edit' 'Search'.

If you select the **In message window** button, all locations where the series of symbols searched for appears in the selected object will be listed line by line in tabular form in the message window. Afterward, the number of locations found will be displayed.

If the report window was not opened, it will be displayed. For each location that is found, the following will be displayed:

- Object name
- Location of the find in the Declaration (Decl) or in the Implementation (Impl) portion of a POU
- Line and network number if any
- The full line in the text editors
- Complete text element in the graphic editors

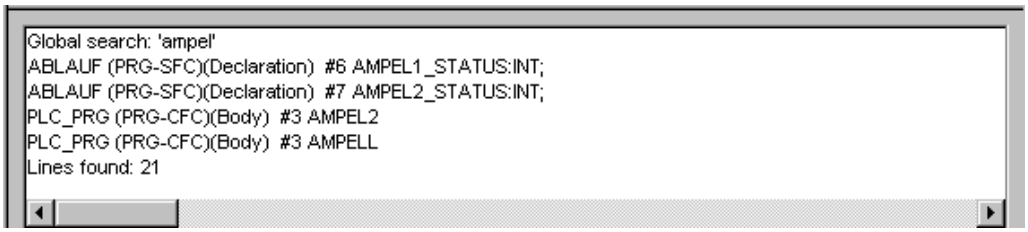


Image 4.42: Message window with search result display

If you double-click the mouse on a line in the message window or press <Enter>, the editor opens with the object loaded. The line concerned in the object is marked. You can jump rapidly between display lines using the function keys <F4> and <Shift>+<F4>.

‘Project‘ ‘Global replace‘

With this command you can search for the location of a text in POUs, data types, or the objects of the global variables and replace this text by another. This is executed in the same way as with **'Project' 'Global Search'** or **'Edit' 'Replace'**. The libraries, however, are not offered for selection and no display in the message window is possible.

Results are displayed in the message window.

‘Project‘ ‘Check‘

A submenu listing the following commands will open:

- Unused Variables
- Overlapping memory areas
- Access conflict
- Multiple writes to output

Each of these functions tests the state of the most recent compilation. The project must therefore have been compiled error-free at least once, before the test can be carried out; if not, the menu items are “greyed out".

Results are displayed in the message window.

Unused Variables: This function searches for variables that have been declared but not used in the program. They are outputted by POU name and line, e.g.: PLC_PRG (4) – var1. Variables in libraries are not examined.

Overlapping memory areas: This function tests whether in allocation of variables via the “AT” declaration overlaps have arisen at specific memory areas. For example, an overlap occurs when allocating the variables “var1 AT %QB21: INT” and “var2 AT %QD5: DWORD” because they both use byte 21. The output then appears as follows:

%QB21 is referenced by the following variables:
PLC_PRG (3): var1 AT %QB21
PLC_PRG (7): var2 AT %QD5

Access conflict: This function in the 'Project' 'Check' menu searches for memory areas which are referenced in more than one task. No distinction is made here between read and write access. The output is for example:

%MB28 is referenced in the following tasks :
Task1 – PLC_PRG (6): %MB28 [read-only access]
Task2 – POU1.ACTION (1) %MB28 [write access]

Multiple writes to output: This function of the 'Project' 'Check' menu searches for memory areas to which a single project gains write access at more than one place. The output then appears as follows:

%QB24 is written to at the following locations:
PLC_PRG (3): %QB24
PLC_PRG.POU1 (8): %QB24

User groups

In **CoDeSys** up to eight user groups with different access rights to the POUs, data types, visualizations, and resources can be set up. Access rights for single objects or

all of them can be established. Only a member of a certain user group can open a project. A member of such a user group must identify himself by means of a password.

The user groups are numbered from 0 to 7, whereby the Group 0 has the administrator rights, i.e. only members of group 0 may determine passwords and access rights for all groups and/or objects.

When a new project is launched, then all passwords are initially empty. Until a password has been set for the 0 group, one enters the project automatically as a member of the 0 group.

If a password for the user group 0 is existing while the project is loaded, then a password will be demanded *for all* groups when the project is opened. For this the following dialog box appears:

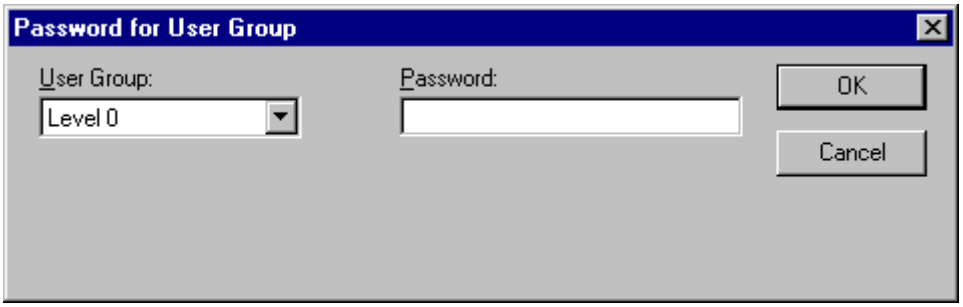


Image 4.43: Dialog box for password entry

In the combobox **User group** on the left side of the dialog box, enter the group to which you belong and enter on the right side the relevant **password**. Press **OK**. If the password does not agree with the saved password, then the message appears:

"The password is not correct."

Only when you have entered the correct password the project can be opened.

With the command 'Passwords for user group' you can assign the passwords, and with 'Object' 'Access rights' you can define the rights for single objects or for all of them.

‘Project‘‘Passwords for user groups‘

With this command you open the dialog box for password assignment for user groups. This command can only be executed by members of group 0. When the command has been given, then the following dialog box appears:

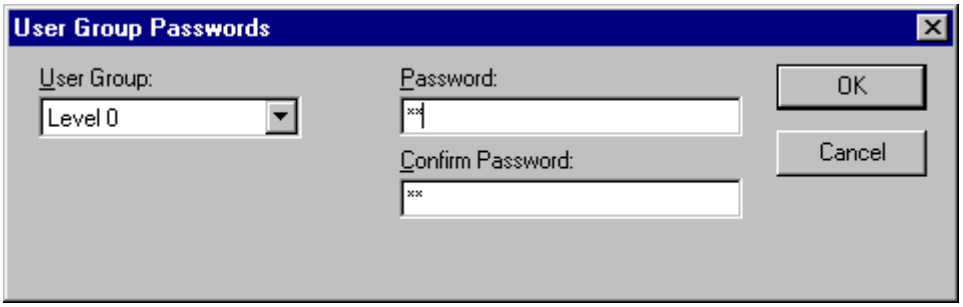


Image 4.44: Dialog box for password assignment

In the left combobox **User group** you can select the group. Enter the desired password for the group in the field **Password**. For each typed character an asterisk (*) appears in the field. You must repeat the same password in the field **Confirm password**. Close the dialog box after each password entry with **OK**. If you get the message:

"The password does not agree with the confirmation",

then you made a typing error during one of the two entries. In this case repeat both entries until the dialog box closes without a message.

Then, if necessary, assign a password for the next group by calling the command again.

Important: If passwords are not assigned to all user groups, a project can be opened by way of a group to which no password was assigned!

Use the command **'Object' 'Access rights'** to assign the rights for single objects or all of them.

4.4 Managing Objects in a Project

Now we shall explain how to work with objects and what help is available to keep track of a project (Folders, Call tree, Cross reference list,..).

Object

POUs, data types, visualizations and the resources global variables, the variable configuration, the Sampling Trace, the PLC Configuration, the Task Configuration, and the Watch and Receipt Manager are all defined as "objects". The folders inserted for structuring the project are partially involved. All objects of a project are in the Object Organizer.

If you hold the mouse pointer for a short time on a POU in the Object Organizer, then the type of the POU (Program, Function or Function block) is shown in a Tooltip. For the global variables the tooltip shows the keyword (VAR_GLOBAL, VAR_CONFIG).

With drag & drop you can shift objects (and also folders, see 'Folder') within an object type. For this, select the object and shift it to the desired spot by holding down the left mouse button. If the shift results in a name collision, the newly introduced element will be uniquely identified by an appended, serial number (e.g. “Object_1”).

Folder

In order to keep track of larger projects you should group your POUs, data types, visualizations, and global variables systematically in folders.

You can set up as many levels of folders as you want. If a plus sign is in front of a closed folder symbol , then this folder contains objects and/or additional folders. With a click on the plus sign the folder is opened and the subordinated objects appear. With a click on the minus (which has replaced the plus sign) the

folder can be closed again. In the context menu you find the commands **'Expand nodes'** and **'Collapse nodes'** with the same functions.

With Drag&Drop you can move the objects as well as the folders within their object type. For this select the object and drag it with pressed left mouse button to the desired position.

You can create more folders with the command 'New folder'.

Note: Folders have no influence on the program, but rather serve only to structure your project clearly.

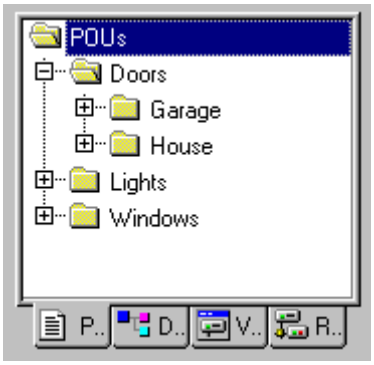


Image 4.45: Example of folders in the Object Organizer

'New Folder'

With this command a new folder is inserted as a structural object. If a folder has been selected, then the new one is created underneath it. Otherwise it is created on the same level. If an action is selected, the new folder will be inserted at the level of the POU to which the action belongs.

The context menu of the Object Organizer which contains this command appears when an object or the object type has been selected and you have pressed the right mouse button or <Shift>+<F10>.

The newly inserted folder initially has the designation 'New Folder'. Observe the following naming convention for folders:

- Folders at the same level in the hierarchy must have distinct names. Folders on different levels can have the same name.
- A folder can not have the same name as an object located on the same level.

If there is already a folder with the name “New Folder” on the same level, each additional one with this name automatically receives an appended, serial number (e.g. “New Folder 1”). Renaming to a name that is already in use is not possible.

'Expand nodes' 'Collapse nodes'

With the command expand the objects are visibly unfolded which are located in the selected object. With Collapse the subordinated objects are no longer shown.

With folders you can open or close them with a double mouse click or by pressing <Enter>.

The context menu of the Object Organizer which contains this command appears when an object or the object type has been selected and you have pressed the right mouse button or <Shift>+<F10>.

'Project' 'Object Delete'

Shortcut: <Delete>

With this command the currently selected object (a POU, a data type, a visualization, or global variables), or a folder with the subordinated objects is removed from the Object Organizer and is thus deleted from the project

For safety you are asked once more for confirmation.

If the editor window of the object was open, then it is automatically closed.

If you delete with the command 'Edit' 'Cut', then the object is parked on the clipboard.

'Project' 'Object Add'

Shortcut: <Insert>

With this command you create a new object. The type of the object (POU, data type, visualization, or global variables) depends upon the selected register card in the Object Organizer. Enter the **Name of the new POU** in the dialog box which appears. Remember that the name of the object may not have already been used.

Take note of the following restrictions:

- The name of a POU can not include any spaces
- A POU can not have the same name as another POU, or a data type.
- A data type can not receive the same name as another data type or a POU.
- A global variable list can not have the same name as another global variable list.
- An action can not have the same name as another action in the same POU.
- A visualization can not have the same name as another visualization.

In all other cases, identical naming is allowed. Thus for example actions belonging to different POUs can have the same name, and a visualization may have the same as a POU.

In the case of a POU, the POU type (program, function or function block) and the language in which it is programmed must also be selected. 'Program' is the default value of **Type of the POU**, while that of **Language of the POU** is that of most recently created POU. If a POU of the function type is created, the desired data type must be entered in the **Return Type** text input field. Here all elementary and defined data types (arrays, structures, enumerations, aliases) are allowed. Input assistance (e.g. via <F2>) can be used.

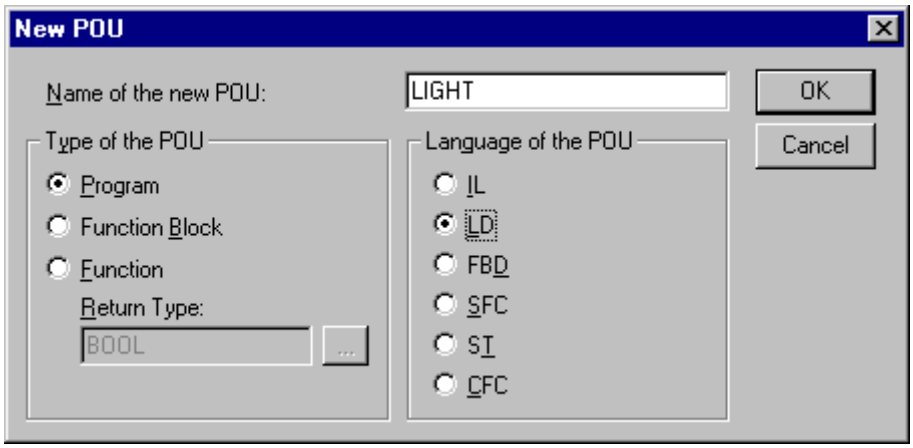


Image 4.46: Dialog for creating a new POU

After pressing **OK**, which is only possible if there is no conflict with the naming conventions described above, the new object is set up in the Object Organizer and the appropriate input window appears.

If the command **'Edit' 'Insert'** is used, the object currently in the clipboard is inserted and no dialog appears. If the name of the inserted object conflicts with the naming conventions (see above), it is made unique by the addition of a serial number appended with a leading underline character (e.g. “Rightturnsig_1”).

If the project is under source control in an **ENI** data base,‘ it may be (depends on the settings in the Project options dialog for 'Project source control') that you will be automatically asked in which data base category you want to handle the new object. In this case the dialog **Properties** will open where you can assign the object to one of the data base object categories.

‘Project‘ ‘Object Rename‘

Shortcut: <Spacebar>

With this command you give a new name to the currently-selected object or folder. Remember that the name of the object may not have already been used.

If the editing window of the object is open, then its title is changed automatically when the name is changed.

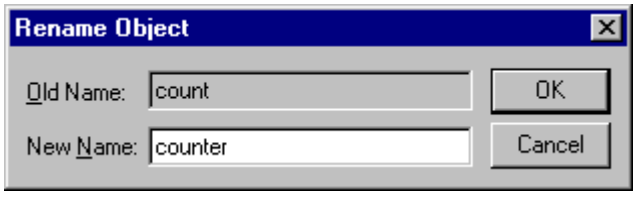


Image 4.47: Dialog box for renaming a POU

‘Project‘ ‘Object Convert‘

This command can only be used with POUs. You can convert POUs from the languages SFC, ST, FBD, LD, and IL into one of the three languages IL, FBD, and LD.

For this the project must be compiled. Choose the language into which you want to convert and give the POU a new name. Remember that the name of the POU may

not have already been used. Then press **OK**, and the new POU is added to your POU list.

The type of processing that occurs during conversion corresponds to that which applies to compilation.

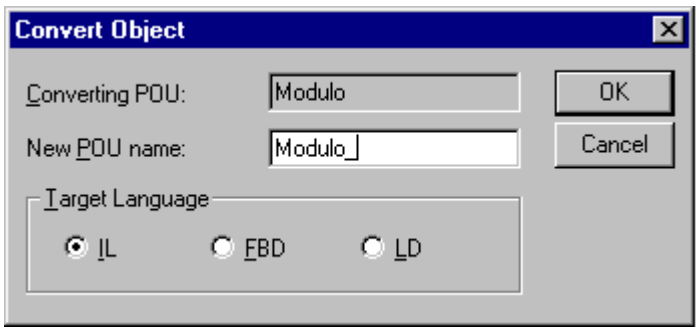


Image 4.48: Dialog box for converting a POU

'Project' 'Object Copy'

With this command a selected object is copied and saved under a new name. Enter the name of the new object in the resulting dialog box. Remember that the name of the object may not have already been used.

If, on the other hand, you used the command **'Edit' 'Copy'**, then the object is parked on the clipboard, and no dialog box appears.

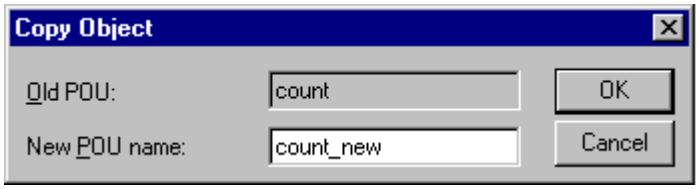


Image 4.49: Dialog box for copying a POU

‘Project‘ ‘Object Open‘

Shortcut: <Enter>

With the command you load a selected object within the Object Organizer into the respective editor. If a window with this object is already open, then it gets a focus, is moved into the foreground and can now be edited.

There are two other ways of opening an object:

- Doubleclick with the mouse on the desired object
- type in the Object Organizer the first letter of the object name. Then a dialog box opens in which all objects of the available object types with this initial letter are shown. Select the desired object and click on the button **Open** in order to load the object in its edit window. This option is supported with the object type Resources only for global variables.

This last possibility is especially useful in projects with many objects.

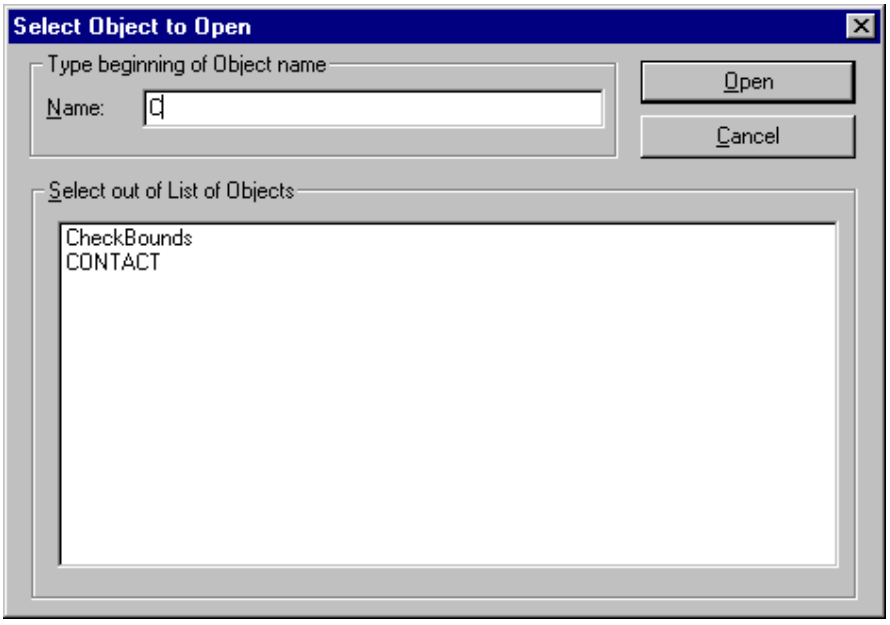


Image 4.50: Dialog box for choosing the object to be opened

'Project‘ ‘Object Access rightsC

With this command you open the dialog box for assigning access rights to the different user groups. The following dialog box appears:

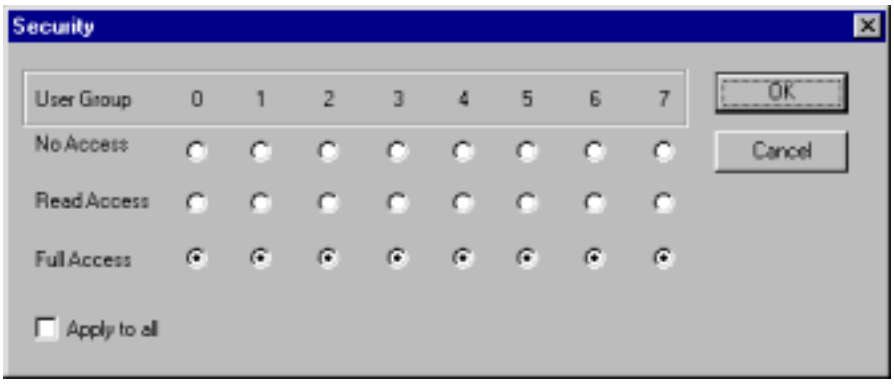


Image 4.51: Dialog box for assigning access rights

Members of the user group 0 can now assign individual access rights for each user group. There are three possible settings:

- **No Access**: the object may not be opened by a member of the user group.
- **Read Access**: the object can be opened for reading by a member of the user group but not changed.
- **Full Access**: the object may be opened and changed by a member of the user group.

The settings refer either to the currently-selected object in the Object Organizer or, if the option **Apply to all** is chosen, to all POUs, data types, visualizations, and resources of the project.

The assignment to a user group takes place when opening the project through a password request if a password was assigned to the user group 0.

'Project' 'Object properties'

This command will open the dialog 'Properties' for that object which is currently marked in the Object organizer.

On the tab **Access rights** you find the same dialog as you get when executing the command 'Project' 'Object Access Rights'.

It depends on the object and the project settings, whether there are additional tabs available where you can define object properties:

- If a global variable list is currently selected in the Object Organizer, then a tab '**Global variable list**' will be available where the parameters concerning the actualization of the list and concerning the data exchange of network variables are defined. The entries can be modified here. This dialog also will be opened if you create a new global variable list by selecting one of the entries in section 'Global Variables' in the Object Organizer and executing the command 'Add Object'.
- If a visualization object is currently selected in the Object Organizer and if the option 'Web visualization' resp. 'Target visualization' is activated in the Target Settings, then a tab '**Visualization**' will be available, where you can choose, whether the object should be part of the Web visualization resp. Target visualization.

If the project is connected to an ENI data base (see Chapter 2.2, 'Project' 'Options' 'Project source control'), then a tab '**Database-connection**' will be available. Here you can display and modify the current assignment of the object to one of the data base categories resp. to the category 'Local'. See for further information Chapter 7: 'What is ENI'.

'Project' 'Data Base Link'

This menu item is only available if you have activated the option 'Use source control (ENI)' in the project options dialog for category 'Project source control. A submenu is attached where you find the following commands for handling the object resp. the project in the currently connected ENI data base:

- **Login** (The user logs in to the ENI Server)

If an object is marked in the Object Organizer and the command **Data Base Link** is executed (from the **context menu**, right mouse button), then the following commands will be available for executing the corresponding data base actions. If the

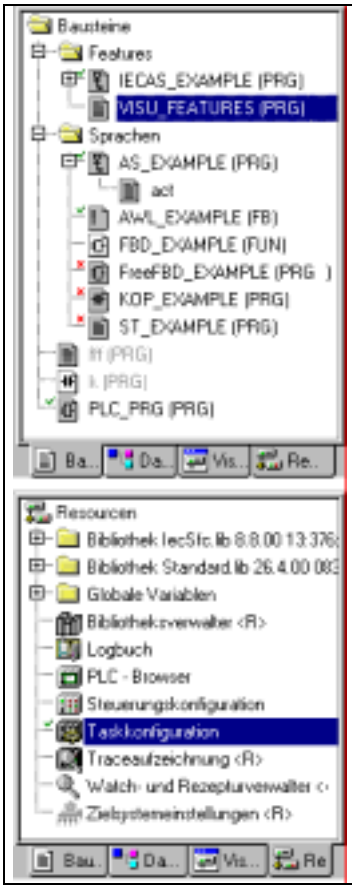
user had not logged in successfully to the ENI Server before, then the dialog 'Data base Login' will open automatically and the chosen command will not been executed until the login was successful:

- Define
- Get Latest Version
- Check Out
- Check In
- Undo Check Out
- Show differences
- Show Version History

If the command 'Data Base Link' in the 'Project' menu is activated, then additional menu items will be available, which concern all objects of the project:

- Multiple Define
- Get All Latest Versions
- Multiple Check Out
- Multiple Check In
- Multiple Undo Check Out
- Project Version History
- Label Version
- Add Shared Objects
- Refresh Status

See in the following how the status of an object resp. its handling in the data base is displayed in the Object Organizer:



	Grey shaded icon: Object is kept in the data base Green check in front of the object name: Object is checked out in the local project. Red cross in front of the object name: Object is currently checked out by another user. <R> behind object name: The object can only be read, but not edited. Please regard: some objects (Task configuration, Sampling Trace, PLC Configuration, Target Settings, Watch- and Receipt Manager) are per default assigned with a <R> as long as they are not checked out. This means that you will not automatically be asked whether the object should be checked out, as soon as you start to edit the object; it not necessarily means that you cannot edit the object. If there is actually no write access then the command 'Check out' will not be available.

Data base command 'Define'

Use this command of the menu 'Project' 'Data Base Link' to define whether the object, which is currently marked in the Object organizer, should be kept in the data base or just locally in the project. A dialog will open, where you can choose one of the two data base categories 'Project' or 'Shared objects', or the category 'Local'.

The icons of all objects which are managed in the data base will be displayed grey-shaded in the Object organizer.

Data base command 'Get Latest Version'

Use this command of the menu 'Project' 'Data Base Link' to copy the current version of the object, which is marked in the Object organizer, from the data base to your project. This will overwrite the local version. In contrast to the 'Check Out' action the object will not be locked for other users in the data base.

Data base command 'Check Out'

Use this command of the menu 'Project' 'Data Base Link' to check out the object, which is marked in the Object organizer, from the data base and by that to lock it for other users.

When executing the command the user will get a dialog 'Check out object'. A comment can be added there which will be stored in the version history of the object in the data base. After the dialog has been closed with OK the checked-out object will be marked with a green check in the object organizer of the local project. For other users it will be appear marked with a red cross and will not be editable by them.

Data base command 'Check In'

Use this command of the menu 'Project' 'Data Base Link' to check in the object, which is marked in the Object organizer, to the data base. Thereby a new version of the object will be created in the data base. The old versions will be kept anyway.

When executing the command the user will get a dialog 'Check in object'. There a comment can be added which will be stored in the version history of the object in the data base. After the dialog has been closed with OK the green check in front of the object name in the Object organizer will be removed.

Data base command 'Undo Check Out'

Use this command of the menu 'Project' 'Data Base Link' to cancel the Checking out of the object which is currently marked in the Object organizer. Thereby also the modifications of the object which have been made locally, will be canceled. No dialog will appear. The unchanged last version of the object will be kept in the data base and it will be accessible again for other users. The red cross in front of the object name in the Object organizer will disappear.

Data base command 'Show Differences'

Use this command of the menu 'Project' 'Data Base Link' to display the object, which is currently opened by the user in CoDeSys, in a window which is divided up in two parts. There the local version, which is currently edited by the local user, will be opposed to the last (actual) version which is kept in the data base. The differences of the versions will be marked like described for the project comparison (see 'Project' 'Compare').

Data base command 'Show Version History'

Use this command of the menu 'Project' 'Data Base Link' to open a dialog 'Version history of <object name> for the object which is currently marked in the Object Organizer. There all versions of the object are listed which have been checked in to the data base or which have been labeled there:

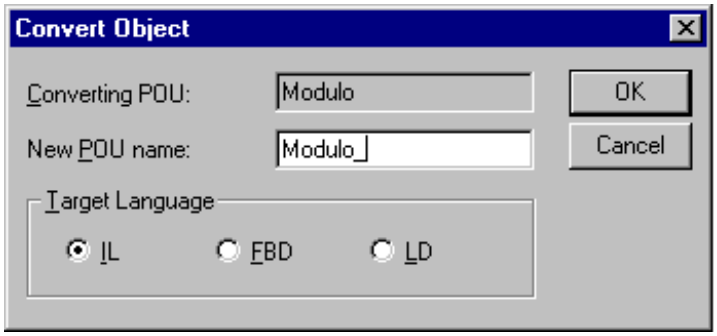


Image 4.52: Dialog Version History

The following information is given:

Version: Data base specific numbering of the versions of the object which have been checked in one after the other. Labeled versions get no version number but are marked by a label-icon.

User: Name of the user, who has executed the check-in or labeling action

Date: Date and time stamp of the action

Action: Type of the action which has been executed. Possible types: 'created' (the object has been checked in to the data base for the first time), 'checked in' (all check-in's of the object excluding the first one) and 'labeled with <label>' (a label has been assigned to this version of the object)

The buttons:

Close: The dialog will be closed.

Display: The version which is currently marked in the table will be opened in a window in CoDeSys. The title bar shows: "ENI: <name of the project in the data base>/<object name>

Details: The dialog 'Details of Version History' will open:

> **File** (name of the project and the object in the data base), **Version** (see above), **Date** (see above), **User** (see above), Comment (Comment which has been inserted when the object has been checked in resp. has been labeled). Use the

buttons **Next** resp. **Previous** to jump to the details window of the next or previous entry in the table in dialog 'Version history of ..'.

Get latest version: The version which is marked in the table will be loaded in CoDeSys and there will overwrite the local version.

Differences: If in the table only one version of an object is marked, then this command will cause a comparison of this version with the latest (actual) data base version. If two versions are marked, then those will be compared. The differences are displayed in a bipartited window like it is done at the project comparison.

Reset version: The version which is marked in the table will be set as latest version. All versions which have been checked in later will be deleted ! This can be useful to restore an earlier status of an object.

Labels only: If this option is activated, then only those versions of the object will be displayed in the table, which are marked by a label.

Selection box below the option 'Labels only': Here you find the names of all users which have executed any data base actions for objects of the current project. Select 'All' or one of the names if you want to get the version history concerning all users or just for a certain one.

Data base command 'Multiple Define'

Use this command of the menu 'Project' 'Data Base Link' to assign several objects at a single blow to a certain data base category. The dialog **'Properties'** will open like described for command 'Define'. Choose the desired category and close the dialog with OK. After that the dialog **'ENI-Selection'** will open, listing all POUs of the project which are considered for the chosen category (Example: if you choose category 'shared objects' then the selection window will only offer the POUs of the Resources tab). The POUs are presented in a tree structure complying to that of the Object Organizer. Select the desired POUs and confirm with OK.

Data base command 'Get All Latest Versions'

Use this command of the menu 'Project' 'Data Base Link' to call the latest version of each object of the currently opened project, which is kept under source control, from the data base. Consider the following:

- If in the meantime additional objects have been stored to the data base project folder, then those will now be added to the local project in CoDeSys.
- If objects have been deleted in the data base in the meantime, those will not be deleted in the local project, but they will automatically get assigned to category 'Local'.
- The latest version of objects of category 'Shared Objects' will only be called, if these objects are already available in the local project. For further information see command 'Get latest version'.

Data base command Multiple Check Out

Use this command of the menu 'Project' 'Data Base Link' to check out several objects at a single blow. For this the dialog **'ENI-Selection'** will open, listing all POUs of the project. Select those which should be checked out and confirm with OK. For further information see command 'Check Out'.

Data base command 'Multiple Check In'

Use this command of the menu 'Project' 'Data Base Link' to check in several objects at a single blow. For this the dialog **'ENI-Selection'** will open, listing all POUs of the project. Select those which should be checked in and confirm with OK. For further information see command 'Check In'.

Data base command 'Multiple Undo Check Out'

Use this command of the menu 'Project' 'Data Base Link' to undo the check out action for several objects at a single blow. For this the dialog **'ENI-Selection'** will open, listing all POUs of the project. Select those for which you want to cancel the check out and confirm with OK. For further information see command 'Undo Check Out'.

Data base command 'Project Version History'

Use this command of the menu 'Project' 'Data Base Link' to view the version history for the currently opened project.But this will only work, if the chosen data base system supports that functionality.

The dialog 'History of <data base project name>' will open. It shows the actions (create, check in, label) which have been performed for the particular objects of the project in a chronological order. The total number of objects is displayed behind **Version history**. The dialog can be handled like described for command 'Show Version History', but regard the following:

- The command 'Reset Version' is only available for single objects.
- The command 'Get latest version' will call all objects of the version of the currently marked entry to the local project ! That means, that the objects in CoDeSys will be overwritten with the older version. But: Local objects, which were not yet part of the project in that older version, will not be removed from the local project !

Data base command 'Label Version'

Use this command of the menu 'Project' 'Data Base Link' to put a "label" on the actual version of each object of a project, so that exactly this project version can be recalled later. A dialog 'Label <data base project name>' will open. Insert a label name (**Label**) (e.g. "Release Version") and optionally a **Comment**. When you confirm with OK, the dialog will close and the label and the action "labeled with <label name>" will appear in the table of the version history, as well in the history

for a single object as in the history of the project. A labeled version of the project does not get a version number, but is just marked with a label icon in the column 'Version'. If the option 'Labels only' is activated in the Version History dialog, then only labeled versions will be listed.

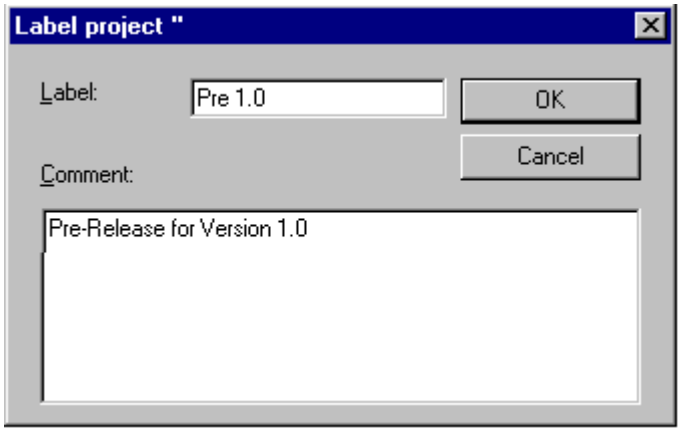


Image 4.53: Dialog for labeling a project version

Data base command 'Add Shared Objects'

Use this command of the menu 'Project' 'Data Base Link' if you explicitely want to add new objects of data base category 'Shared Objects' to the locally opened project in CoDeSys. For objects of category 'Project Objects' this is not necessary, because the command 'Get (all) latest version(s)' automatically calls all objects which are found in the data base project folder, even if there are some which not yet available in the local project. But for objects of category 'Shared Objects' in this case just those objects will be called which are already available in the local project.

So execute the command 'Add Shared Objects' to open the dialog 'Browse ENI'. A list in the right part of the window shows all objects which are available in the data base folder which is currently selected in the list on the left side. Choose the desired object and press OK or do a doubleclick on the entry to insert the object to the currently opened CoDeSys project.

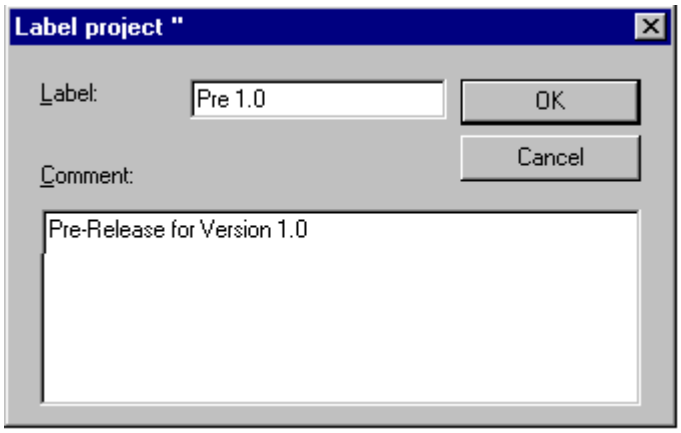


Image 4.54: Dialog 'Browse ENI' for adding 'Shared objects' to the local project

Data base command 'Refresh Status'

Use this command of the menu 'Project' 'Data Base Link' to update the display in the Object Organizer, so that you can see the actual status of the objects concerning the source control of the project.

Login

Use this command of the menu 'Project' 'Data Base Link' to open the dialog 'Login' where you can enter the access data for the ENI data base via the ENI Server. The access data also have to be defined in the ENI Server (ENI Admin, User Management) and – depending on the currently used data base – also in the user management of the data base. After the command has been executed, first the Login dialog for category 'Project objects' will open.

The following items are displayed:

Data base: project objects

Host: address of the computer where the ENI Server is running (must match with the entry in field 'TCP/IP address' in the project options dialog for 'Project source control').

Project: Name of the data base project (must match with the entry in field 'Project name' in the project options dialog for 'Project source control'/category 'Project Objects').

Credentials: Insert **User name** and **Password**.

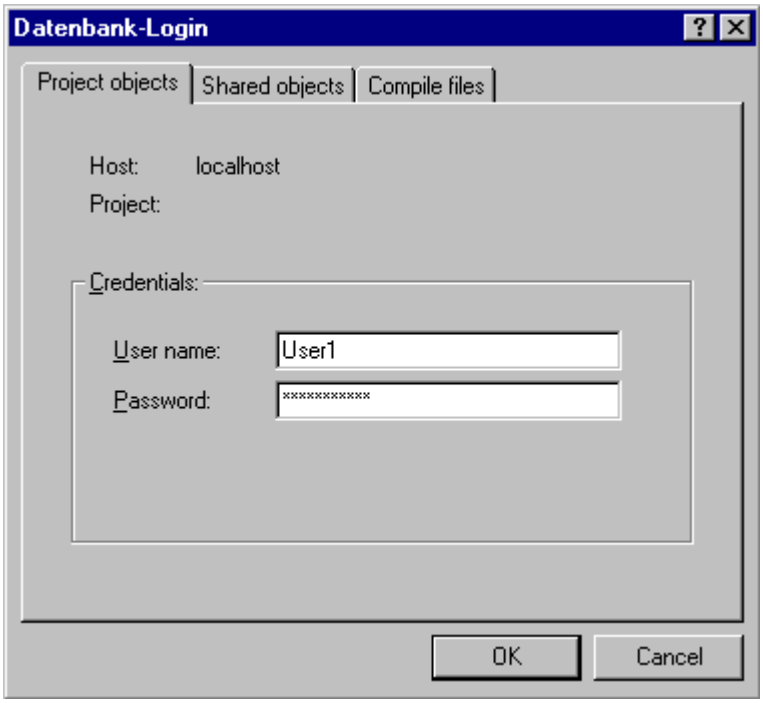


Image 4.55: ENI Login Dialog

When option **Use as default** for this project is activated, then the above entered access data will automatically be used for any further communication between the actual CoDeSys project and the data base concerning objects of the actual category.

Press OK to confirm the settings. The dialog will be closed and automatically the Login dialog for 'Shared objects' will open. Enter the access data in the same way as described for the 'Project objects' and confirm with OK. Do the same in the third Login dialog which will be opened for category 'Compile files'.

The Login dialog will always open as soon as you try to access the data base before having logged in successfully like described above.

Note: If you want to save the access data with the project, activate option 'Save ENI credentials' in the project options, category 'Load & Save'.

'Project' 'Add Action'

This command is used to generate an action allocated to a selected block in the Object Organizer. One selects the name of the action in the dialog which appears and also the language in which the action should be implemented.

The new action is placed under your block in the Object Organiser. A plus sign appears in front of the block. A simple mouse click on the plus sign causes the action objects to appear and a minus sign appears in front of the block. Renewed clicking on the minus sign causes the actions to disappear and the plus sign appears again. This can also be achieved over the context menu commands **'Expand Node'** and **'Collapse Node'.**

'Project' 'Open Instance'

With this command it is possible to open and display the instance of the function block which is selected in the Object Organizer. In the same manner, a double click on the function block in the Object Organizer gives access to a selection dialog in which the instances of the function block as well as the implementation are listed. Select here the desired instance or the implementation and confirm using OK. The desired item is then displayed in a window.

Attention: If you want to view instances, you first have to log in ! (The project has been compiled with no errors and downloaded to the PLC with **'Online' 'Login'**).

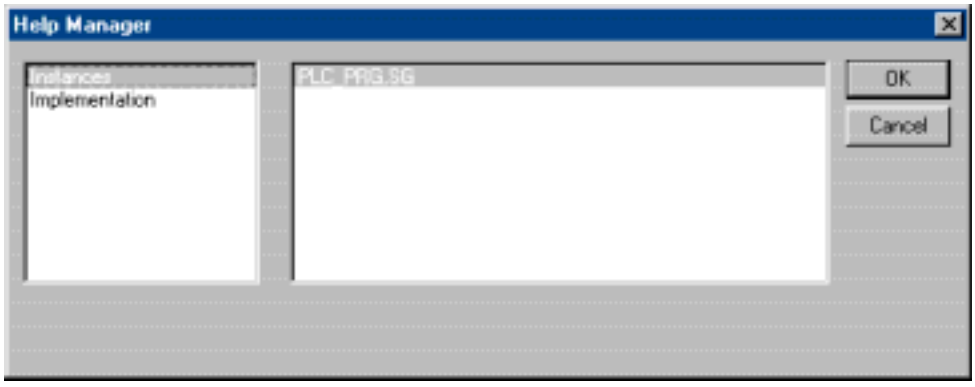


Image 4.56: Dialog for opening an instance

‘Project‘ ‘Show Call Tree‘

With this command you open a window which shows the call tree of the object chosen in the Object Organizer. For this the project must be compiled (see **'Rebuild all'**). The call tree contains both calls for POUs and references to data types.

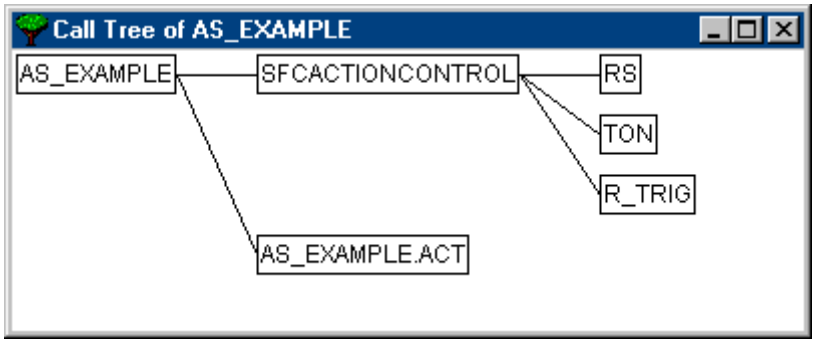


Image 4.57: Example of a call tree

‘Projekt’ ‘Show Cross Reference‘

With this command you open a dialog box which makes possible the output of all application points for a variable, address, or a POU. For this the project must be compiled (see **'Project' 'Build').**

Choose first the **category** 'Variable', 'Address', or 'POU' and then enter the **name** of the desired element. To obtain all elements of the entered category enter a “*” in Name.

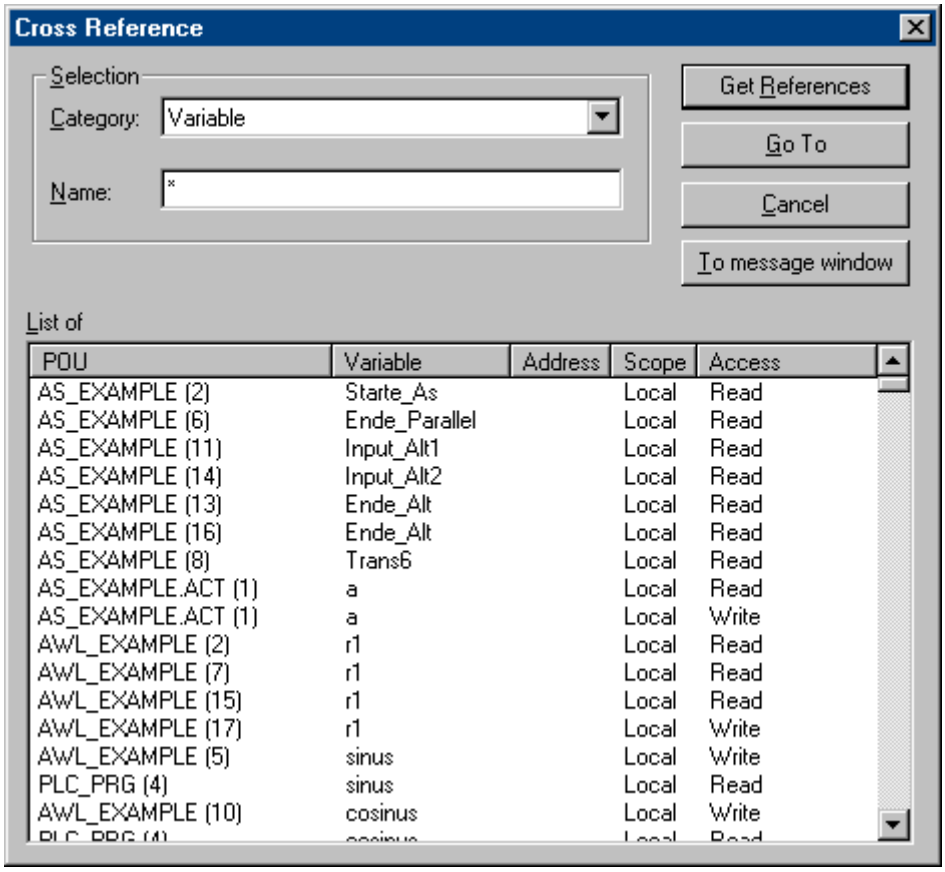

Cross Reference
Selection
Category: Variable
Name: *
Get References
Go To
Cancel
To message window
List of
POU | Variable | Address | Scope | Access
AS_EXAMPLE (2) | Starte_As | | Local | Read
AS_EXAMPLE (6) | Ende_Parallel | | Local | Read
AS_EXAMPLE (11) | Input_Alt1 | | Local | Read
AS_EXAMPLE (14) | Input_Alt2 | | Local | Read
AS_EXAMPLE (13) | Ende_Alt | | Local | Read
AS_EXAMPLE (16) | Ende_Alt | | Local | Read
AS_EXAMPLE (8) | Trans6 | | Local | Read
AS_EXAMPLE.ACT (1) | a | | Local | Read
AS_EXAMPLE.ACT (1) | a | | Local | Write
AWL_EXAMPLE (2) | r1 | | Local | Read
AWL_EXAMPLE (7) | r1 | | Local | Read
AWL_EXAMPLE (15) | r1 | | Local | Read
AWL_EXAMPLE (17) | r1 | | Local | Write
AWL_EXAMPLE (5) | sinus | | Local | Write
PLC_PRG (4) | sinus | | Local | Read
AWL_EXAMPLE (10) | cosinus | | Local | Write

Image 4.58: Dialog box and example of a cross reference list

By clicking on the button **Cross References** you get the list of all application points. Along with the POU and the line or network number, the variable name and the address binding, if any, are specified. The Domain space shows whether this is a local or a global variable; the Access column shows whether the variable is to be accessed for ‚reading' or ‚writing' at the current location.

When you select a line of the cross reference list and press the button **Go To** or doubleclick on the line, then the POU is shown in its editor at the corresponding point. In this way you can jump to all application points without a time-consuming search.

In order to make processing easier, you can use the **Send to message window** button to bring the current cross reference list into the message window and from there change to the respective POU.

4.5 General Editing Functions

You can use the following commands in all editors and some of them in the Object Organizer. The commands are located under the menu item **'Edit'** and in the context menu that is opened with the right mouse button.

If the IntelliPoint-Software is installed on the computer, CoDeSys supports all functions of the MS IntelliMouse. In all editors with zoom functionality: To magnify press the <Strg> key while rolling the wheel of the mouse, to reduce roll backwards while the <Strg> key is pressed.

‘Edit‘ ‘Undo‘

Shortcut: <Ctrl>+<Z>

This command undoes the action which was most recently executed in the currently-open editor window or in the Object Organizer; repeated use undoes all actions back to the time that the window was opened. This applies to all actions in the editors for POUs, data types, visualizations and global variables and in the Object Organizer.

With 'Edit' 'Redo' you can restore an action which you have undone.

Note: The commands **Undo** and **Redo** apply to the current window. Each window carries its own action list. If you want to undo actions in several windows, then you must activate the corresponding window. When undoing or redoing in the Object Organizer the focus must lie here.

‘Edit‘ ‘Redo‘

Shortcut: <Ctrl>+<Y>

With the command in the currently-open editor window or in the Object Organizer you can restore an action you have undone (**'Edit' 'Undo'**).

As often as you have previously executed the command **'Undo'** , you can also carry out the command **'Redo'**.

Note: The commands **'Undo'** and **'Redo'** apply to the current window. Each window carries its own action list. If you want to undo actions in several windows, then you must activate the corresponding window. When undoing or redoing in the Object Manager must lie there.

'Edit' 'Cut'

Symbol:

Shortcut: <Ctrl>+<X> or <Shift>+<Delete>

This command transfers the current selection from the editor to the clipboard. The selection is removed from the editor.

In the Object Organizer this similarly applies to the selected object, whereby not all objects can be deleted, e.g. the PLC Configuration.

Remember that not all editors support the cut command, and that its use can be limited in some editors.

The form of the selection depends upon the respective editor:

In the text editors IL, ST, and declarations the selection is a list of characters.

In the FBD and LD editors the choice is a number of networks which are indicated by a dotted rectangle in the network number field or a box with all preceding lines, boxes, and operands.

In the SFC editor the selection is a part of a series of steps surrounded by a dotted rectangle.

In order to paste the content of the clipboard you use the command 'Edit' 'Paste'. In the SFC editor you can also use the commands 'Extras' 'Insert parallel branch (right)' or 'Extras' 'Paste after'.

In order to copy a selection onto the clipboard without deleting it, use the command 'Edit' 'Copy'.

In order to remove a selected area without changing the clipboard, use the command 'Edit' 'Delete'.

'Edit' 'Copy'

Symbol: **Shortcut: <Ctrl>+<C>**

This command copies the current selection from the editor to the clipboard. This does not change the contents of the editor window.

With the Object Organizer this similarly applies to the selected object, whereby not all objects can be copied, e.g. the PLC Configuration.

Remember that not all editors support copying and that it can be limited with some editors.

For the type of selection the same rules apply as with **'Edit' 'Cut'**.

The form of the selection depends upon the respective editor:

In the text editors (IL, ST, and declarations) the selection is a list of characters.

In the FBD and LD editors the choice is a number of networks which are indicated by a dotted rectangle in the network number field or a box with all preceding lines, boxes, and operands.

In the SFC editor the selection is a part of a series of steps surrounded by a dotted rectangle.

In order to paste the content of the clipboard you use the command 'Edit' 'Paste'. In the SFC editor you can also use the commands 'Extras' 'Insert parallel branch (right)' or 'Extras' 'Paste after'.

In order to delete a selected area and simultaneously put it on the clipboard, use the command '**Edit**' '**Cut**'.

'Edit' 'Paste'

Symbol: **Shortcut: <Ctrl>+<V>**

Pastes the content of the clipboard onto the current position in the editor window. In the graphic editors the command can only be executed when a correct structure results from the insertion.

With the Object Organizer the object is pasted from the clipboard.

Remember that pasting is not supported by all editors and that its use can be limited in some editors.

The current position can be defined differently according to the type of editor:

With the text editors (IL, ST Declarations) the current position is that of the blinking cursor (a vertical line) which you place by clicking with the mouse).

In the FBD and LD editors the current position is the first network with a dotted rectangle in the network number area. The contents of the clipboard are inserted in front of this network. If a partial structure has been copied, then it is inserted in front of the selected element.

In the SFC editor the current position is determined the selection which is surrounded by a dotted rectangle. Depending upon the selection and the contents of the clipboard, these contents are inserted either in front of the selection or into a new branch (parallel or alternative) to the left of the selection.

In SFC the commands 'Extras' 'Insert parallel branch (right)' or 'Extras' 'Paste after' can be used in order to insert the contents of the clipboard.

In order to copy a selection onto the clipboard without deleting it, use the command '**Edit**' '**Copy**'.

In order to remove a selected area without changing the clipboard, use the command 'Edit' 'Delete'.

'Edit' 'Delete'

Shortcut: <Del>

Deletes the selected area from the editor window. This does not change the contents of the clipboard.

In the Object Organizer this applies likewise to the selected object, whereby not all objects can be deleted, e.g. the PLC Configuration.

For the type of selection the same rules apply as with **'Edit' 'Cut'**.

The form of the selection depends upon the respective editor:

In the text editors (IL, ST, and declarations) the selection is a list of characters.

In the FBD and LD editors the selection is a number of networks which are highlighted with a dotted rectangle in the network number field.

In the SFC editor the selection is a part of a series of steps surrounded by a dotted rectangle.

In the library manager the selection is the currently selected library name.

In order to delete a selected area and simultaneously put it on the clipboard, use the command **'Edit' 'Cut'**.

'Edit' 'Find'

Symbol:

With this command you search for a certain text passage in the current editor window. The Find dialog box opens. It remains opens until the button **Cancel** is pressed.

In the field **Find what** you can enter the series of characters you are looking for.

In addition, you can decide whether the text you are looking for **Match whole word only** or not, or also whether **Match case** is to be considered, and whether the search should proceed **Up** or **Down** starting from the current cursor position.

The button **Find next** starts the search which begins at the selected position and continues in the chosen search direction. If the text passages is found, then it is highlighted. If the passage is not found, then a message announces this. The search can be repeated several times in succession until the beginning or the end of the contents of the editor window has been reached. In the CFC editor the geometrical order of the elements will be regarded, the search will run from the left upper corner of the window to the right upper corner. Please regard that FBD POUs are processed from the right to the left !

Image 4.59: Find dialog box

'Edit' 'Find next'

Symbol: **Shortcut: <F3>**

With this command you execute a search with the same parameters as with the most recent action **'Edit' 'Find'.** Please regard that FBD POUs are processed from the right to the left !

'Edit' 'Replace'

With this command you search for a certain passage just as with the command **'Edit' 'Find',** and replace it with another. After you have chosen the command the dialog box for find and replace appears. This dialog box remains open until the button **Cancel** or **Close** is pressed.

In the field behind **Find** automatically that string will be inserted which you have marked before in the editor. You also can enter the search string manually. Pressing button **Replace** will replace the current selection with the text in the field **Replace with**. Use the button **Find Next** to get to the next passage where the string is found. Please regard, that FBD POUs are processed from the right to the left !

The button **Replace all** replaces every occurrence of the text in the field **Find next** after the current position with the text in the field **Replace with**. At the end of the procedure a message announces how many replacements were made.

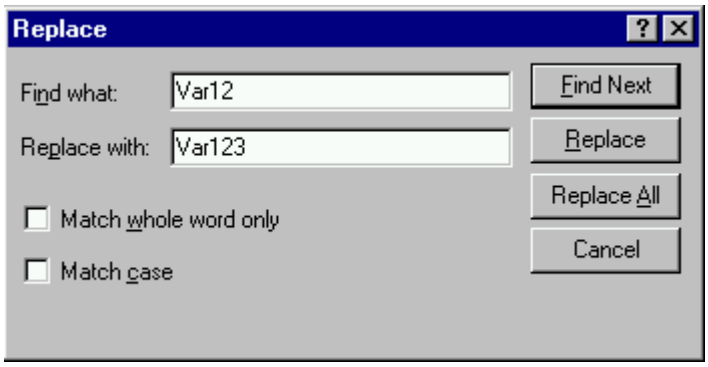


Image 4.60: Dialog box for find and replace

'Edit' 'Input Assistant'

Shortcut: <F2>

This command provides a dialog box for choosing possible inputs at the current cursor position in the editor window. In the left column choose the desired input category, select the desired entry in the right column, and confirm your choice with **OK**. This inserts your choice at this position.

The categories offered depend upon the current cursor position in the editor window, i.e. upon that which can be entered at this point (e.g. variables, operators, POUs, conversions, etc.).

If the option **With arguments** is active, then when the selected element is inserted, the arguments to be transferred are specified with it, for example: function block fu1 selected, which defines the input variable var_in: fu1(var_in:=);

Insertion of function func1, which uses var1 and var2 as transfer parameters: func1(var1,var2)

It is basically possible to switch between structured and unstructured display of the available elements. This occurs through activation/deactivation of the **Structured Display** option.

Note: For inserting identifiers you also can use the Intellisense functionality.

- Unstructured Display:

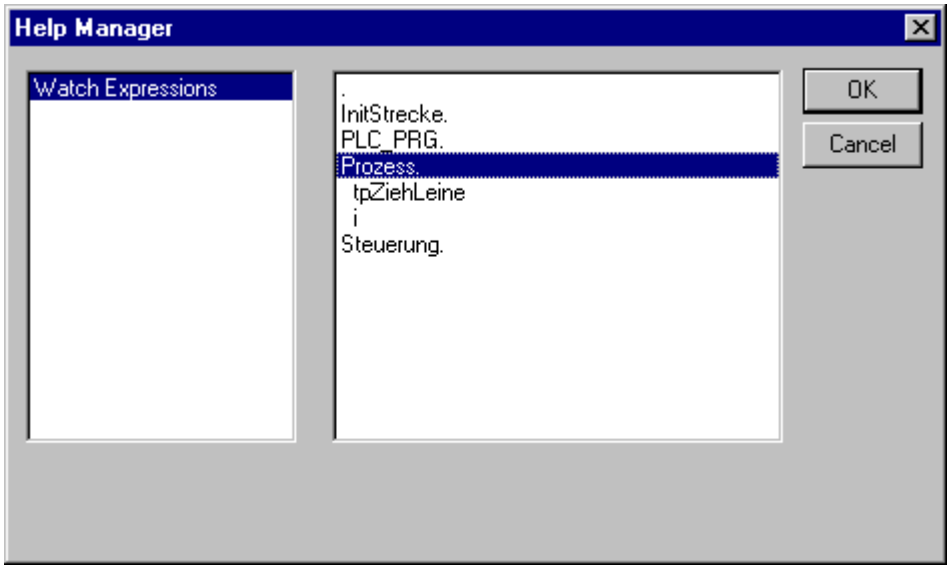


Image 4.61: Dialog for unstructured input assistance

The POUs, variables or data types in each category are simply sorted linearly in alphabetical order.

At various places (e.g. in the Watch List), multi-stage variable names are required. In that event, the Input Assistant dialog displays a list of all POUs as well as a single point for the global variables. After each POU name there is a point. If a POU is selected by doubleclick or by pressing <Enter>, a list of the variables belonging to it opens. If instances and data types are present, it is possible to open further levels in the hierarchy display. **OK** transfers the most recently selected variable.

- Structured Display:

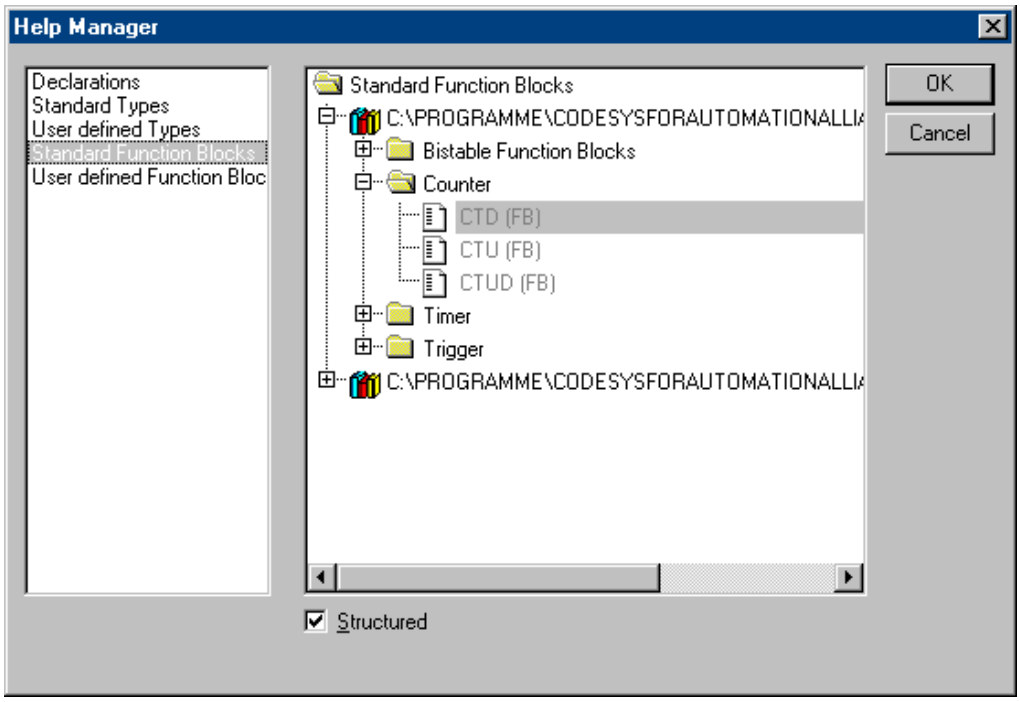


Image 4.62: Dialog for structured input assistance

If **Structured display** is selected, the POUs, variables or data types will be sorted hierarchically. This is possible for standard programs, standard functions, standard function blocks, defined programs, defined functions, defined function blocks, global variables, local variables, defined types, watch variables. The visual and hierarchical display corresponds to that of the Object Organizer; if elements in a library are referred to, these are inserted in alphabetical order at the very top and the pertinent hierarchy is displayed as in the Library Manager.

The **in- and output variables** of function blocks which are declared as local or global variables are listed in the category 'Local Variables' or 'Global Variables' under the instance name (e.g. Inst_TP ET, Inst_TP IN,...). To get there, select the instance name (e.g. Inst_TP) and confirm with **OK**.

If the **instance of a function block** is selected here, the option **With arguments** may be selected. In the text languages ST and IL as well as during task configuration, the instance name and the input parameters of the function block are then inserted.

For example, if Inst (DeklarationInst: TON;) is selected, the following is inserted:

Inst(IN:= ,PT:=)

If the option is not selected, only the instance name will be inserted. In the graphical languages or in the Watch window, only the instance name is generally inserted.

Components of **structures** are displayed in an analog fashion to function block instances.

For **enumerations**, the individual enumeration values are listed under the enumeration type. The order is: enumerations from libraries, enumerations from data types, local enumerations from POUs.

The general rule is that lines containing sub-objects are not selectable (except instances, see above), but can only have their hierarchy display expanded or contracted by one level, as for multi-stage variable names.

If Input Assistant is invoked in the Watch and Receipt Manager or in the selection of trace variables in the trace configuration dialog, it is possible to make a **multiple selection**. When the <Shift> key is pressed, you can select a range of variables; when the <Ctrl> key is pressed you can select many individual variables. The selected variables are so marked. If, during range selection lines are selected that do not contain valid variables (e.g. POU names), these lines will not be included in the selection. When individual selections are made, such lines can not be marked.

In the **watch window** and in **trace configuration** it is possible to transfer structures, arrays or instances from the Input Assistant dialog. As a double click with the mouse button is associated with the extension or contraction of the element’s hierarchy display, selection in these cases can only be confirmed by **OK**.

Thereafter, the selected variables are inserted line by line in the watch window, that is each selected variable is written on a separate line. In the case of trace variables, each variable is inserted in a separate line of the trace variables list.

If the maximum number of trace variables, 20, is exceeded during insertion of the selected variables, the error message „A maximum of 20 variables is allowed“ appears. Further selected variables are then not inserted in the list.

Note: Some entries (e.g. Global Variables) are only updated in the Input Assistant dialog after compilation.

‘Edit’ ’Declare Variable’

Shortcut: <Shift>+<F2>

This command opens the dialog for the declaration of a variable. This dialog also opens automatically when the option 'Project' 'Options' 'Editor' 'Autodeclaration' is switched on and when a new undefined variable is used the declaration editor.

‘Edit‘ ‘Next error'

Shortcut: <F4>

After the incorrect compilation of a project this command can show the next error. The corresponding editor window is activated and the incorrect place is selected. At the same time in the message window the corresponding error message is shown.

‘Edit‘ ‘Previous error'

Shortcut: <Shift>+<F4>

After the incorrect compilation of a project this command shows the previous error. The corresponding editor window is activated and the incorrect place is selected. At the same time in the message window the corresponding error message is shown.

‘Edit’ ’Macros’

This menu item leads to a list of all macros, which are defined for the project. (For info on generating macros see 'Project' 'Options' 'Macros'). When an executable macro is selected the dialog 'Process Macro'. The name of the macro and the currently active command line are displayed. The button Cancel can be used to stop the processing of the macro. In that event the processing of the current command will be finished anyway. Then an appropriate message is displayed in the message window and in the log during Online operation: "<Macro>: Execution interrupted by user“.

Macros can be executed offline and online, but in each case only those commandes are executed which are available in the respective mode.

4.6 General Online Functions

The available online commands are assembled under the menu item **'Online'**. The execution of some of the commands depends upon the active editor.

The online commands become available only after logging in.

Thanks to 'Online Change' functionality you have the possibility of making changes to programs on the running controller. See in this connection 'Online' 'Log-in'.

'Online' 'Login'

Symbol: **Shortcut: <Alt>+<F8>**

This command combines the programming system with the PLC (or starts the simulation program) and changes into the online mode.

If the current project has not been compiled since opening or since the last modification, then it is compiled now (as with **‘Project‘ ‘Build)**. If errors occur during compilation, then **CoDeSys** does not change into Online mode.

If the current project was changed on the controller since the last download, but not closed, and if the last download information was not deleted with the command 'Project' 'Clear all', then after the command 'Login' a dialog opens with the question: „The program has been changed. Load changes? (**Online Change**)“. By answering **Yes** you confirm that, on log-in, the modified portions of the project are to be loaded onto the controller. **No** results in a log-in without the changes made since the last download being loaded onto the controller. **Cancel** cancels the command. <Load all> causes the entire project to be reloaded onto the controller.

After a successful login all online functions are available (if the corresponding settings in 'Project' 'Options' category 'Build' have been entered). The current values are monitored for all visible variable declarations.

Use the 'Online' 'Logout' command to change from online back to offline mode.

If the system reports

Error:
"The selected controller profile does not match that of the target system..."

Check that the target system entered in the target system settings (Resources) matches the parameters entered in **'Online' 'Communications parameters'**.

Error:
„Communication error. Log-out has occurred“

Check whether the controller is running. Check whether the parameters entered in **'Online' 'Communications parameters'** match those of your controller. In particular, you should check whether the correct port has been entered and whether the baud rates in the controller and the programming system match. If the gateway server is used, check whether the correct channel is set.

Error:
"The program has been modified! Should the new program be loaded?"

The project which is open in the editor is incompatible with the program currently found in the PLC (or with the Simulation Mode program being run). Monitoring and debugging is therefore not possible. You can either choose "No," logout, and open the right project, or use "Yes" to load the current project in the PLC.

Message:
„The program has been changed. Load changes? (ONLINE CHANGE)“.

The project is running on the controller. The target system supports 'Online Change' and the project has been altered on the controller with respect to the most recent download or the most recent Online Change. You may now decide whether these changes should be loaded with the controller program running or whether the

command should be cancelled. You can also, however, load the entire compiled code by selecting the **Load all** button.

'Online' 'Logout'

Symbol: **Shortcut <Strg>+<F8>**

The connection to the PLC is broken, or, the Simulation Mode program is ended and is shifted to the offline mode.

Use the 'Online' 'Login' command to change to the online mode.

'Online' 'Download'

This command loads the compiled project in the PLC.

If you use C-Code generation, then prior to the download, the C-Compiler is called up, which creates the download file. If this is not the case, then the download file is created during the compiling.

If you use C-Code generation, then prior to the download, the C-Compiler is called up, which creates the download file. If this is not the case, then the download file is created during the compilation.

Download information is saved in a file called **<projectname>0000000ar.ri** , which is used during Online Change to compare the current program with the one most recently loaded onto the controller, so that only changed program components are reloaded. This file is erased by the command **'Project' 'Clear all'**.

'Online' 'Run'

Symbol: **Shortcut: <F5>**

This command starts the program in the PLC or in Simulation Mode.

This command can be executed immediately after the 'Online' 'Download' command, or after the user program in the PLC has been ended with the 'Online' 'Stop' command, or when the user program is at a break point, or when the 'Online' 'Single Cycle' command has been executed.

'Online' 'Stop'

Symbol: **Shortcut <Shift>+<F8>**

Stops the execution of the program in the PLC or in Simulation Mode between two cycles.

Use the 'Online' 'Run' command to continue the program.

'Online' 'Reset'

This command resets – with exception of the retain variables (VAR RETAIN) - all variables to that specific value, with which they have got initialized (also those variables which have been declared as VAR PERSISTENT !). If you have initialized the variables with a specific value, then this command will reset the variables to the initialized value. All other variables are set at a standard initialization (for example, integers at 0). As a precautionary measure, **CoDeSys** asks you to confirm your decision before all of the variables are overwritten. The situation is that which occurs

in the event of a power failure or by turning the controller off, then on (warm restart) while the program is running.

Use the 'Online' 'Run' command to restart the program.

See also ‘Online’ ‘Reset (original)' und ‘Online’ ‘Reset (cold)'.

‘Online’ ‘Reset (cold)’

This command resets, with exception of the persistent variables (VAR PERSISTENT) all variables, also retain variables !, back to their initialization values. The situation is that wich occurs at the start of a program which has been downloaded just before to the PLC. Only persistent variables retain the value that they had before the reset.

Use the 'Online' 'Run' command to restart the program.

‘Online’ ‘Reset (original)’

This command resets all variables including the remanent ones (VAR RETAIN and VAR PERSISTENT) to their initialization values and erases the user program on the controller. The controller is returned to its original state. See in this connection also 'Online' 'Reset' and 'Online' 'Cold Reset'

'Online' 'Toggle Breakpoint'

Symbol: **Shortcut: <F9>**

This command sets a breakpoint in the present position in the active window. If a breakpoint has already been set in the present position, that breakpoint will be removed.

The position at which a breakpoint can be set depends on the language in which the POU in the active window is written.

In the Text Editors (IL, ST), the breakpoint is set at the line where the cursor is located, if this line is a breakpoint position (recognizable by the dark-gray color of the line number field). You can also click on the line number field to set or remove a breakpoint in the text editors.

In FBD and LD, the breakpoint is set at the currently selected network. In order to set or remove a breakpoint in the FBD or LD Editor, you can also click on the network number field.

In SFC, the breakpoint is set at the currently selected step. In SFC you can also use <Shift> with a doubleclick to set or remove a breakpoint.

If a breakpoint has been set, then the line number field or the network number field or the step will be displayed with a light-blue background color.

If a breakpoint is reached while the program is running, the program will stop, and the corresponding field will be displayed in a red background color. In order to continue the program, use the **'Online' 'Run'**, **'Online' 'Step in'**, or **'Online' 'Step Over'** commands.

You can also use the Breakpoint dialog box to set or remove breakpoints.

‘Online‘ ‘Breakpoint Dialog Box‘

This command opens a dialog box to edit breakpoints throughout the entire project. The dialog box also displays all breakpoints presently set.

In order to set a breakpoint, choose a POU in the **POU** combobox and the line or the network in the **Location** combobox where you would like to set the breakpoint; then press the **Add** button. The breakpoint will be added to the list.

In order to delete a breakpoint, highlight the breakpoint to be deleted from the list of the set breakpoints and press the **Delete** button**.**

The **Delete All** button can be used to delete all the breakpoints.

In order to go to the location in the editor where a certain breakpoint was set, highlight the respective breakpoint from the list of set breakpoints and press the **Go to** button**.**

To set or delete breakpoints, you can also use the **'Online' 'Toggle Breakpoint'** command.

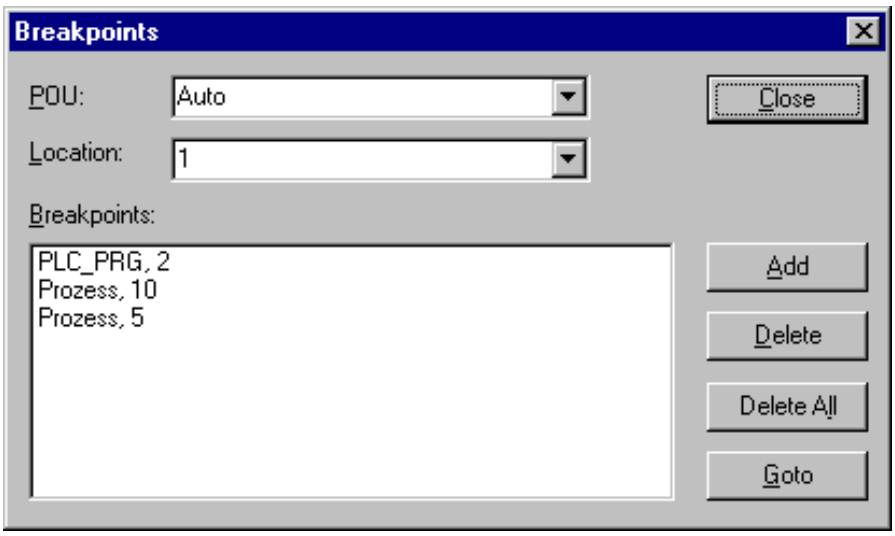


Image 4.63: Breakpoint Editing Dialog Box

‘Online‘ ‘Step over‘

Symbol: **Shortcut: <F10>**

This command causes a single step to execute. If a POU is called, the program stops after its execution. In SFC a complete action is executed.

If the present instruction is the call-up of a function or of a function block, then the function or function block will be executed completely. Use the **'Online' 'Step In'** command, in order to move to the first instruction of a called function or function block.

If the last instruction has been reached, then the program will go on to the next instruction in the POU.

'Online' 'Step in'

Shortcut: <F8>

A single step is executed. The program is stopped before the first instruction of a called POU.

If necessary, there will be a changeover to an open POU.

If the present position is a call-up of a function or of a function block, then the command will proceed on to the first instruction in the called POU.

In all other situations, the command will function exactly as **'Online' 'Step Over'**.

'Online' 'Single Cycle'

Shortcut: <Ctrl>+<F5>

This command executes a single PLC Cycle and stops after this cycle.

This command can be repeated continuously in order to proceed in single cycles.

The Single Cycle ends when the **'Online' 'Run'** command is executed.

'Online' 'Write values'

Shortcut: <Ctrl>+<F7>

With this command, one or more variables are set – one time only! – to user defined values at the beginning of a cycle. (see **'Online' 'Force values'** for setting permanently)

The values of all single-element variables can be changed, so long as they are also visible in Monitoring.

Before the command 'Write values' can be executed, a variable value must be ready to be written:

1. Define the values for the Writelist:

- For non-boolean variables a double mouse click in performed on the line in which a variable is declared, or the variable is marked and the <Enter> key is pressed. The dialog box 'Write variable <x>' then appears, in which the value to be written to the variable can be entered.

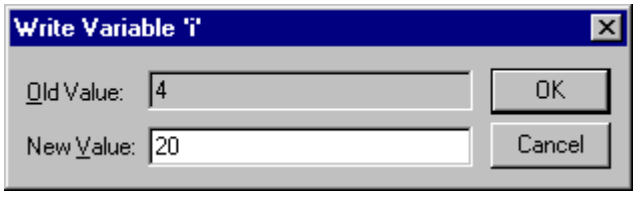


Bild 4.64: Dialog for writing of variables

- For boolean variables, the value is toggled (switched between TRUE and FALSE, with no other value allowed) by double-clicking on the line in which the variable is declared; no dialog appears.

The value set for Writing is displayed in brackets and in turquoise colour behind the former value of the variable. e.g. a=0 <:=34>.

Hint: Exception: In the FBD and LD Editor the value is shown turquoise without brackets next to the variable name.

Set the values for as many variables as you like.

The values entered to be written to variables can also be corrected or deleted in the same manner. This is likewise possible in the 'Online' 'Write/Force dialog' (see below).

The values to be written that were previously noticed are saved in a **writelist (Watchlist)**, where they remain until they are actually written, deleted or transferred to a forcelist by the command 'Force values'.

2. Write the values:

The command to Write Values can be found at two places::

- Command 'Write Values' in the menu 'Online'.
- Button 'Write Values' in the dialog 'Editing the writelist and the forcelist'.

When the command 'Write values' is executed, all the values contained in the writelist are written, once only , to the appropriate variables in the controller at the beginning of the cycle, then deleted from the writelist. (If the command '**Force values**' is executed, the variables in question are also deleted from the writelist, and transferred to the forcelist!)

Note: In the sequential function chart language (SFC), the individual values from which a transition expression is assembled cannot be changed with 'Write values'. This is due to the fact that in monitoring the 'Total value' of the expression, not the values of the individual variables are displayed (e.g. "a AND b" is only displayed as TRUE if both variables actually have the value TRUE).

In FBD, on the other hand, only the first variable in an expression, used for example as input to a function block, is monitored. Thus a 'Write values' command is only possible for this variable.

'Online' 'Force values'

Shortcut: <F7>

With this command, one or more variables are permanently set (**see 'Online' 'Write values'** for setting only once at the beginning of a cycle) to user-defined values. The setting occurs in the run-time system, both at the beginning and at the end of the cycle.

The time sequence in one cycle: 1.Read inputs, 2. Force values 3. Process code, 4. Force values 5. Write outputs.

The function remains active until it is explicitly suspended by the user (command 'Online' 'Release force') or the programming system is logged-out.

For setting the new values, a **writelist** is first created, just as described under 'Online' 'Write values'. The variables contained in the writelist are accordingly marked in Monitoring. The writelist is transferred to a **forcelist** as soon as the command 'Online' 'Force values' is executed. It is possible that an active forcelist already exists, in which case it is updated as required. The writelist is then emptied and the new values displayed in red as 'forced'. Modifications of the forcelist will be transferred to the program with the next 'Force values' command.

Note: The forcelist is created at the first forcing of the variables contained in the writelist, while the writelist existed prior to the first writing of the variables that it contains.

The command for forcing a variable, which means that it will be entered into the forcelist can be found at the following places:

- Command 'Force Values' in the menu ‘Online'.
- Button 'Force Values' in the dialog 'Editing the writelist and the forcelist'.

Note: In the sequential function chart language, the individual values from which a transition expression is assembled cannot be changed with 'Force values'. This is due to the fact that in monitoring the 'Total value' of the expression, not the values of the individual variables are displayed (e.g. "a AND b" is only displayed as TRUE if both variables actually have the value TRUE).

In FBD, on the other hand, only the first variable in an expression, used for example as input to a function block, is monitored. Thus a 'Force values' command is only possible for this variable.

‘Online’ ‘Release force'

Shortcut: <Shift>+<F7>

This command ends the forcing of variable values in the controller. The variable values change again in the normal way.

Forced variables can be recognized in Monitoring by the red color in which their values are displayed. You can delete the whole forcelist, but you can also mark single variables for which the forcing should be released.

To delete the whole forcelist, which means to release force **for all variables**, choose one of the following ways:

- Command 'Release Force' in menu 'Online'.
- Button 'Release Force' in dialog 'Editing the writelist and the forcelist'
- Delete the whole forcelist using the command ‘Release Force’ in the dialog ‘Remove Write-/Forcelist’. This dialog opens if you choose the command ‘Release Force’ while also a writelist exists.

To release force only **for single variables** you have to mark these variable first. Do this in one ways described in the following. After that the chosen variables are marked with an turquoise extension <Release Force>:

- A double mouse click on a line, in which a non boolean variable is declared, opens the dialog ‘Write variable <x>’. Press button <Release Force for this variable> .

- Repeat double mouse clicks on a line in which a boolean variable is declared to toggle to the display <Release Force> at the end of the line.
- In the menu ‘Online’ open the Write/Force-Dialog and delete the value in the edit field of the column ‘Forced value’.

When for all desired variables the setting “<Release Force>” is shown in the declaration window, choose the command **‘Force’** to transfer the modifications of the forcelist to the program.

If the current writelist (see 'Online' 'Write Values') is not empty while you execute the command ‘Release Force’, the dialog 'Remove Write-/Forcelist’ will be opened. There the user has to decide whether he just wants to **Release Force** or additionally wants to **Remove the writelist** or if he wants to remove both lists.

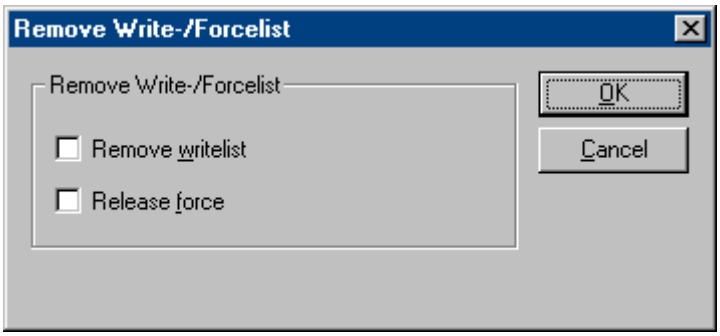


Image 4.65: Dialog for removing Write-/Forcelists

‘Online’ ‘Write/Force' Dialog’

Shortcut: <Ctrl>+<Shift>+<F7>

This command leads to a dialog which displays in two registers the current writelist (**Watchlist**) and forcelist (**Forcelist**). Each variable name and the value to be written to it or forced on it are displayed in a table.

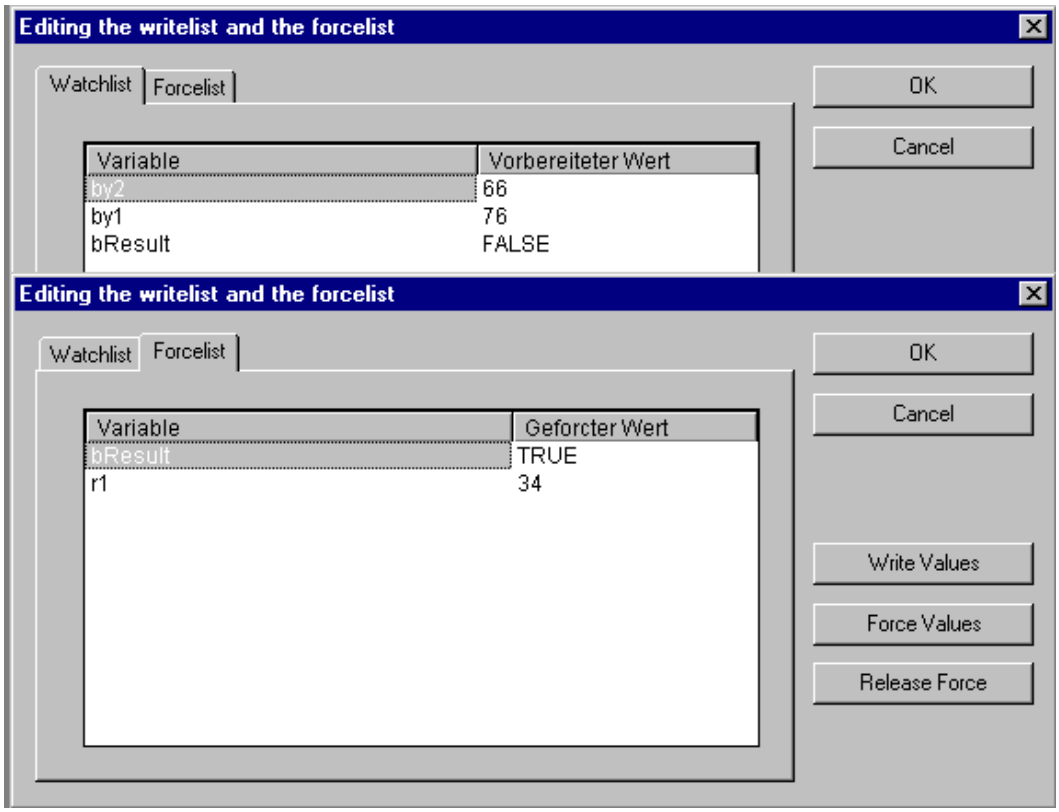


Image 4.66: Dialog for editing the writelist and the forcelist

The variables reach the watchlist via the commands 'Online' 'Write Values' and are transferred to the forcelist by the command 'Online' 'Force Values'. The values can be edited here in the „Prepared Value“ or „Forced Value“ columns by clicking the mouse on an entry to open an editor field. If the entry is not type-consistent, an error message is displayed. If a value is deleted, it means that the entry is deleted from the writelist or the variable is noticed for suspension of forcing as soon as the dialog is closed with any other command than **Cancel**.

The following commands, corresponding to those in the Online menu, are available via buttons:

Force Values: All entries in the current writelist are transferred to the forcelist, that is the values of the variables in the controller are forced. All variables marked with 'Release Force' are no longer forced. The dialog is then closed.

Write Values: All entries in the current writelist are written once only to the corresponding variables in the controller. The dialog is then closed.

Release Force: All entries in the forcelist will be deleted or, if a writelist is present, the dialog “Delete write-/forcelist” comes up, in which the user must decide whether he only wants to **release forcing** or **discard the writelist**, or both. The dialog will close at that point, or after the selection dialog is closed as the case may be.

‘Online‘ ‘Show Call Stack‘

You can run this command when the Simulation Mode stops at a breakpoint. You will be given a dialog box with a list of the POU Call Stack.

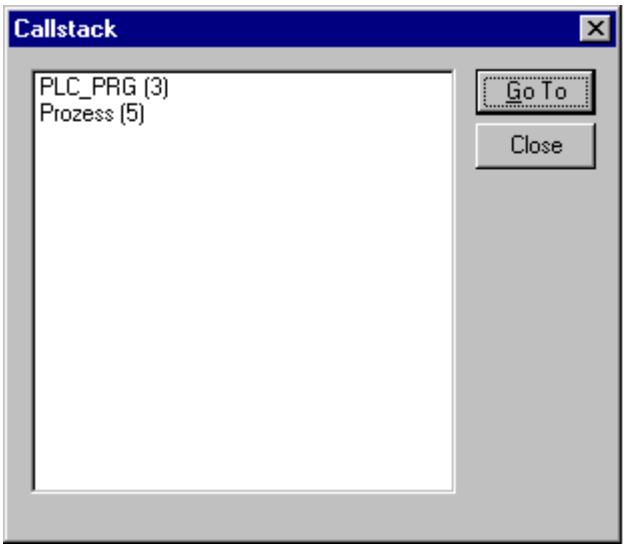


Image 4.67: Example of a Call Stack

The first POU is always PLC_PRG, because this is where the executing begins.

The last POU is always the POU being executed.

After you have selected a POU and have pressed the **Go to** button, the selected POU is loaded in its editor, and it will display the line or network being processed.

'Online' 'Flow Control'"

If you have selected the **flow control**, then a check(✓) will appear in front of the menu item. Following this, every line or every network will be marked which was executed in the last PLC Cycle.

The line number field or the network number field of the lines or networks which just run will be displayed in green. An additional field is added in the IL-Editor in which the present contents of the accumulator are displayed. In the graphic editors for the Function Block Diagram and Ladder Diagram, an additional field will be inserted in all connecting lines not transporting any Boolean values. When these Out- and Inputs are verified, then the value that is transported over the connecting line will be shown in this field. Connecting lines that transport only Boolean values will be shaded blue when they transport TRUE. This enables constant monitoring of the information flow.

‘Online‘ ‘Simulation‘

If **Simulation Mode** is chosen, then a check(✓) will appear in front of the menu item.

In the simulation mode, the user program runs on the same PC under Windows. This mode is used to test the project. The communication between the PC and Simulation Mode uses the Windows Message mechanism.

If the program is not in simulation mode, then the program will run on the PLC. The communication between the PC and the PLC typically runs over the serial interface.

The status of this flag is stored with the project.

'Online' 'Communication Parameters'

You are offered a special dialog for setting communication parameters when the communication between the local PC and the run-time system is running over a gateway server in your system. (If the OPC or DDE server is used, the same communications parameters must be entered in its configuration).

Before the handling of the dialog will be described, see first the

Principle of a gateway system

Let us examine the principle of the gateway system before explaining the operation of the dialog:

A gateway server can be used to allow your local PC to communicate with one or more run-time systems. The setting concerning which run-time systems can be addressed, which is specifically configured for each gateway server, and the connection to the desired gateway server, is made on the local PC. Here it is possible that both the gateway server and the run-time system(s) can run together on the local PC. If we are dealing with a gateway server which is running on another PC we must ensure that it has been started there. If you are selecting a locally installed gateway server, it automatically starts when you log onto the target run-time system. You can recognise this through the appearance of a CoDeSys symbol on the bottom right in the task bar. This symbol lights up as long as you are connected to the run-time system over the gateway. The menu points **Info** and **Finish** are obtained by clicking with the right mousekey on the symbol. **Finish** is used to switch off the gateway.

See the scheme shown below presenting a gateway system:

PC_local is your local PC, **PC_x** is another PC, which gateway addresses. **PC_gateway** is the PC on which the gateway server is installed, **PC_PLC1** through to **PC_PLC4** are PCs on which the run-time systems are running. The diagram shows the modules as separated but it is fully possible for the Gateway server and / or run-time systems to be installed together on the local PC.

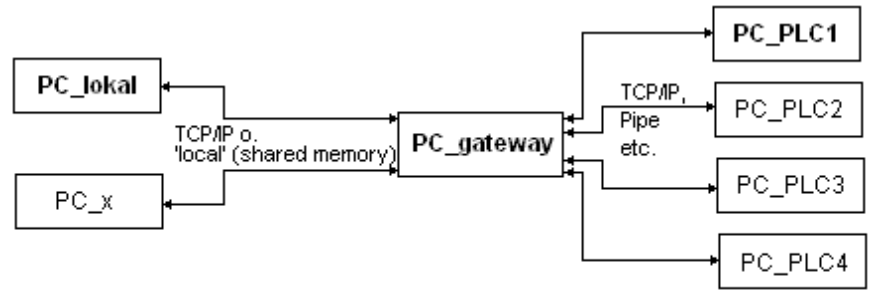


Image 4.68: Example of a Gateway server system

Important: Please note that a connection to gateway is only possible over TCP/IP so make sure that your PC is configured appropriately!

The connections from gateway to the various run-time computers can, on the other hand, run over different protocols (TCP/IP, Pipe, etc.).

Let us now return to the

Communications parameters dialog on the local PC:

This dialog is used to select a gateway server for the communicatio with a PLC. Furtheron there can be set up channels for a gateway server which is installed on the local PC so that these channels can be used by other computers which are part of the network.

The current settings can be called up at any time using the button **Update**.

The dialog will appear as follows if the communications parameters have already been configured according to the example shown above:

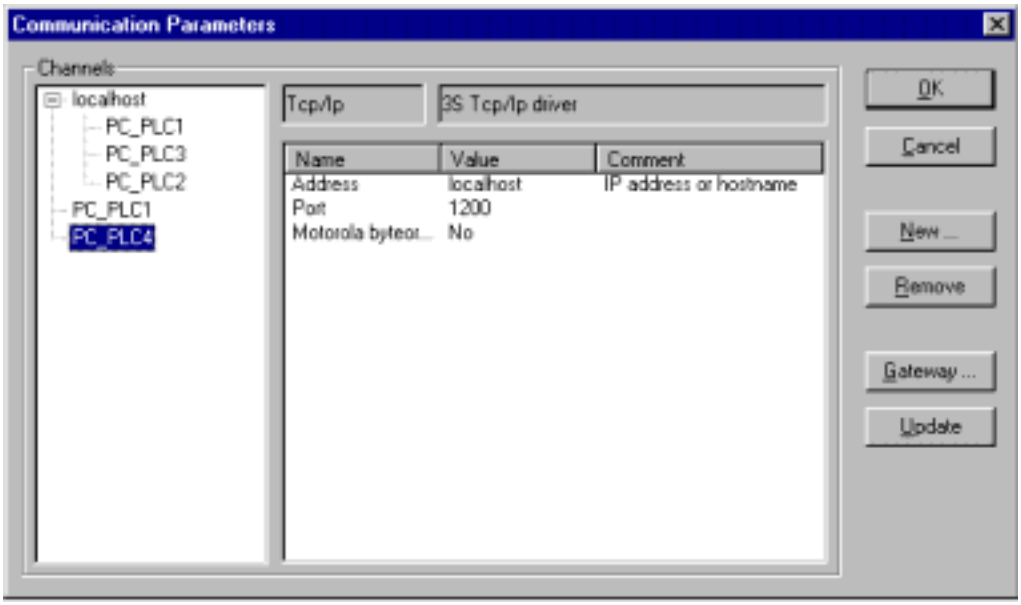


Image 4.69: Dialog for setting the gateway communications parameters, example

The heading **Channels** lists two categories of connections:

On the one hand all of the connections are shown which are installed on the currently connected gateway server called 'localhost'. Here the address or the name of this gateway is located on the upper position behind the minus sign, which in our example is running on the local computer. The appropriate address 'localhost' corresponds in the normal case to the IP address 127.0.0.1 of the local computer (PC_local). Below, indented to the right, are three addresses of run-time computers which the gateway channels are set-up to (PC_PLC1 to 3). They could have been configured both from the local PC or from the other PCs (PC_x) which are or were connected to the gateway server.

The second category of the channels describes includes all connections to the gateway which can be set up from your local PC, over this configuration dialog for example. They create the ”branch” which leads from the minus sign directly below to PC_PLC1 and PC_PLC4. These channel addresses do not necessarily have to be known yet at the gateway. For PC_PLC4 in the example described above, the configuration parameters are stored locally in the project but they will first be known

to the gateway the next time log-in to the run-time system occurs. This has already occurred for PC_PLC1 since the associated gateway address has appeared as an additional ”sub-branch” to the ”channel tree”.

In the central part of the dialog one finds the designation, in each case, of the left selected channel and the associated parameter under **Name**, **Value** and **Comment**.

Let us now switch to

Setting up communication parameters:

1. Setting up the desired gateway server and channel

To define the connection to the desired gateway server we open the dialog 'Communication Parameters Gateway' by pressing the button **Gateway**.

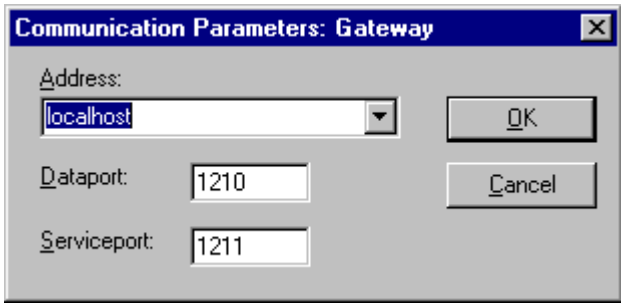


Bild 4.70: Example dialog, definition of the local connection to the gateway

Here you can enter and/or edit the following:

- The type of **connection** from your computer to the computer on which the gateway server that you want to use is running. If the gateway server is running on the local computer, connection via shared memory („local“) or via TCP/IP is possible; if connection to a different computer is needed, only TCP/IP can be used.
- The **address** of the computer, on which the gateway server that you want to use is running: IP address or the appropriate symbolic name such as e.g. localhost. On initial setup, the standard 'localhost' is offered as the computer name (address), which means that the locally installed gateway would be accessed. The name 'localhost' is set to be identical to the local IP address 127.0.0.1 in most cases, but you may in some cases have to enter this directly into the Address field. If you want to access a gateway server on another computer, you must replace 'localhost' with its name or IP address.
- The **password** for the selected gateway server, if it is on a remote computer. If it is incorrectly entered, or not entered at all, an error message appears. Note in this connection: you can give the locally installed gateway server a password with the following procedure: click with the right mouse button on the gateway symbol in the lower right portion of the toolbar and select „Change password“. A dialog comes up for changing or entering a password. If you access the gateway server locally any password that is entered will not be asked for.
- The computer’s **port** on which the gateway server that you wish to use is running, as a rule the correct value for the selected gateway is already given.

If the dialog is closed with **OK**, the corresponding entry (computer address) appears in the **Channels** field at the top of the 'Communication parameters' dialog, and below it the channels available on this gateway server.

2. Setting up the desired channel on the selected gateway server:

Now select one of the channels by clicking on an entry with the mouse. The corresponding parameters will then be shown in the table. If no connection can be established to the selected gateway address — possibly because it has not been started or the address is incorrect — the phrase 'not connected' appears in brackets after the address and a message 'No gateway with these settings could be found' appears. In this connection, see below for further information under 'Quick check in the event of unsuccessful connection attempt to gateway'.

Once the desired channel is set up, close the dialog using **OK**. The settings are saved with the project.

3. Setting up a new channel for the local gateway server:

You can set up new channels for the currently connected gateway server, which are then available for establishing further connection from the server, a connection to a controller for example. The options that you have in this regard depend on the particular choice of the number of device drivers installed on your computer.

Press the **New** button. The dialog 'Communication Parameters: New Channel' comes up:

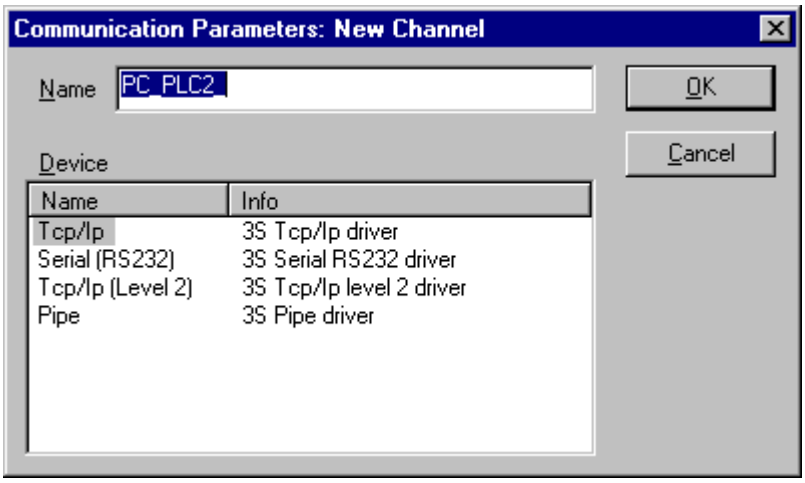


Bild 4.71:Example dialog, installing a new channel

The input field **Name** automatically contains the name used for the last inputted channel. If no channel has yet been defined, the current gateway name will be offered, followed by an underline character , e.g. 'localhost_'. You can edit the channel name at this point. The channel name is purely informative, it does not have to be a unique name but it is recommended to use one.

The device drivers available on the gateway computer are listed in the table under **Device**. In the **Name** column, select by mouse click one of the available drivers; the corresponding comment, if any, appears in the **Info** column.

If you close the '...New Channel' dialog with **OK**, the newly defined channel appears in the 'Communication Parameters' dialog as a new entry in **Channels** at the lowest position under the minus sign. So far, it is only stored locally in the project (see above). At this point you can edit the **Value** column (see tips below). Now confirm the entered parameters with **OK**, thus leaving the 'Communication Parameters' dialog.

In order for the newly entered gateway channel and its parameters to also be known to the gateway server xy, and thus also to make it available to other computers that access this gateway xy, you must log into the run-time system. If you then re-open the 'Online' 'Communication parameters' dialog, the new channel appears in the „channel tree“, not only in its previous position but also indented under the address or name of the gateway server xy. This indicates that it is known to the network. You can now open the Communication Parameter dialog on a computer other than the local one, select gateway xy and use its new channel.

If a communications error occurs when logging in, it is possible that the interface cannot be opened (e.g. COM1 for a serial connection) possibly because it is being used by another device. It is also possible that the controller is not running.

The parameters for a channel already known by the gateway server can no longer be edited in the configuration dialog. The parameter fields appear grey. You can, however, delete the connection as long as it is not active.

Important: Please not that the deletion of a channel is not reversible. It occurs at the moment that you press on the button **Remove**!

Tips for editing the parameters in the communications parameters dialogue:

You can only edit the text fields in the column **Value**.

Select a text field with the mouse, and get into the editing mode by double clicking or by pressing the space bar. The text input is finished by pressing the <Enter> key.

You can use <Tabulator> or <Shift> + <Tabulator> to jump to the next or the previous switching or editing possibility.

To edit numerical values it is possible with the arrow keys or the Page Up/Down keys to change the value by one or ten units respectively. A double click with the mouse also changes the value by increasing by one unit. A typing check is installed for numerical values: <Ctrl> + <Home> or <Ctrl> + <End> deliver the lowest or the highest value respectively for the possible input values for the type of parameter in question.

Quick check in the event of an unsuccessful attempt to set up communication to the gateway server on another computer (You get the message „not connected“ in the Communication Parameters dialog behind the gateway server address in the field Channels):

- Has the gateway server been started (the three-color symbol appears in the bottom right portion of the toolbar) ?

- Is the IP address that you entered in the 'Gateway: Communication Parameters' dialog really that of the computer on which the gateway is running ? (use „ping“ to check)
- Is the TCP/IP connection working locally? The error may possibly lie with TCP/IP.

‘Online’ ‘Sourcecode download’

This command loads the source code for the project into the controller system. This is not to be confused with the Code that is created when the project is compiled! You can enter the options that apply to Download (time, size) in the 'Project' 'Options' 'Sourcedownload' dialog.

‘Online’ 'Create bootproject’

With this command, the compiled project is set up on the controller in such a way that the controller can load it automatically when restarted. Storage of the boot project occurs differently depending on the target system. For example, on 386 systems three files are created: **default.prg** contains the project code, **default.chk** contains the code’s checksum, **default.sts** contains the controller status after restart (start/stop).

The command 'Online' 'Create bootproject' is also available in **offline mode** if the project has been build without errors. In this case the following files are created in the projects directory: **<projektname>.prg** for the bootproject code, and **projektname>.chk** for the checksum. These files can be renamed as necessary and then be copied to a PLC.

Note: If the project option **Implicit at create boot project** (category Sourcedownload) is activated, then the selected sources will be loaded automatically into the controller on the command 'Online' 'Create boot project'.

‘Online’ 'Write file to controller’

This command is used for loading any desired file onto the controller. It opens the dialog for 'Write file to controller' in which you can select the desired file.

After the dialog is closed using the 'Open' button, the file is loaded into the controller and stored there under the same name. The loading process is accompanied by a progress dialog.

With the command **'Online' 'Load file from controller'** you can retrieve a file previously loaded on the controller.

‘Online’ 'Load file from controller’

With this command, you can retrieve a file previously loaded into the controller using 'Online' 'Write file to controller'. You receive the 'Load file from controller' dialog. Under Filename, provide the name of the desired file, and in the selection window enter the directory on your computer into which it is to be loaded as soon as the dialog is closed with the „Save“ button.

4.7 Log

The log stores in chronological order actions that occur during an Online session. For this purpose a binary log file (*.log) is set up. Afterward, the user can store excerpts from the appropriate project log in an external log.

The log window can be opened in either Offline or Online mode and can thus serve as a direct monitor online.

'Window' 'Log'

To open, select the menu item 'Window' 'Log'.

In the log window, the filename of the currently displayed log appears after **Log:**. If this is the log of the current project, the word "(Internal)" will be displayed.

Registered entries are displayed in the log window. The newest entry always appears at the bottom.

Only actions belonging to categories that have been activated in the 'Filter' field of the menu 'Project' 'Options' 'Log' will be displayed.

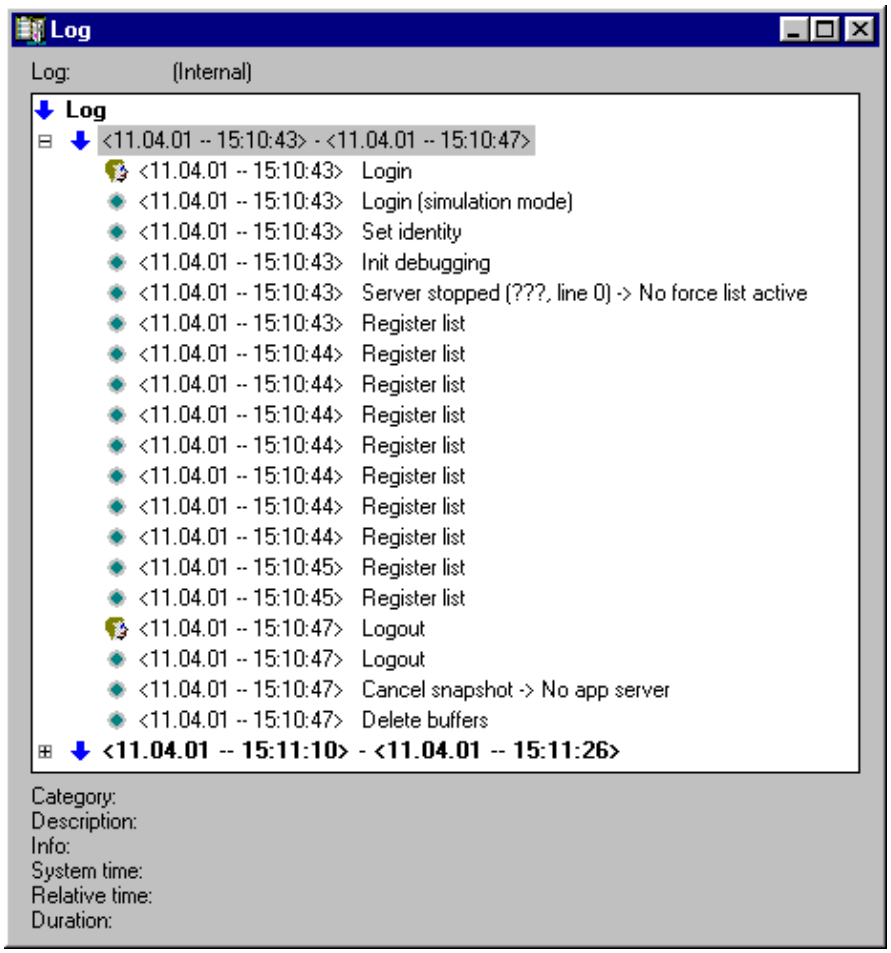


Bild 4.72:Log window

Available information concerning the currently selected entry is displayed below the log window:

Category: The category to which the particular log entry belongs. The following four categories are possible:

- User action: The user has carried out an Online action (typically from the Online menu).
- Internal action: An internal action has been executed in the Online layer (e.g. Delete Buffers or Init Debugging).
- Status change: The status of the runtime system has changed (e.g. from Running to Break, if a breakpoint is reached).
- Exception: An exception has occurred, e.g. a communication error.

Description: The type of action. User actions have the same names as their corresponding menu commands; all other actions are in English and have the same name as the corresponding `OnlineXXX()` function.

Info: This field contains a description of an error that may have occurred during an action. The field is empty if no error has occurred.

System time: The system time at which the action began, to the nearest second.

Relative time: The time measured from the beginning of the Online session, to the nearest millisecond.

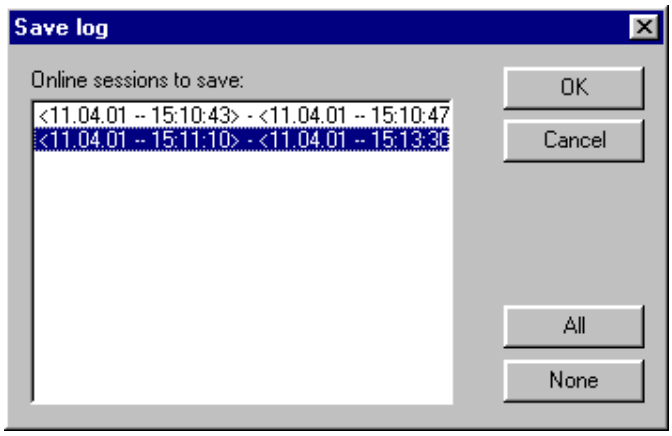
Duration: Duration of the action in milliseconds.

Menu Log

When the log window has the input focus, the menu option **Log** appears in the menu bar instead of the items 'Extras' and 'Options'.

The menu includes the following items:

Load…	An external log file *.log can be loaded and displayed using the standard file open dialog. The log that is present in the project will not be overwritten by the command. If the log window is closed and later opened again, or a new Online session is started then the version that is loaded will again be replaced by the project log.
Save…	This menu item can only be selected if the project log is currently displayed. It allows an excerpt of the project log to be stored in an external file. For that, the following dialog will be displayed, in which the Online sessions to be stored can be selected:



After successful selection, the standard dialog for storing a file opens ('Save Log').

Display Project Log

This command can only be selected if an external log is currently displayed. It switches the display back to the project log.

Storing the project log

Regardless of whether or not the log is stored in an external file (see above), the project log is automatically stored in a binary file entitled <projectname>.log. If a different path is not explicitly given in the 'Project' 'Options' 'Log' dialog, the file is stored in the same directory as that in which the project is stored.

The maximum number of Online sessions to be stored can be entered in the 'Project' 'Options' 'Log' dialog. If this number is exceeded during recording, the oldest session is deleted to make room for the newest one.

4.8 Window set up

Under the **'Window'** menu item you will find all commands for managing the windows. There are commands both for the automatic set up of your window as well as for opening the library manager and for changing between open windows. At the end of the menu you will find a list of all open windows in the sequence they were opened. You can switch to the desired window by clicking the mouse on the relevant entry. A check will appear in front of the active window.

'Window' 'Tile Horizontal'

With this command you can arrange all the windows horizontally in the work area so that they do not overlap and will fill the entire work area.

'Window' 'Tile Vertical'

With this command you can arrange all the windows vertically in the work area so that they do not overlap and will fill the entire work area.

'Window' 'Cascade'

With this command you can arrange all the windows in the work area in a cascading fashion, one behind another.

'Window' 'Arrange Symbols'

With this command you can arrange all of the minimized windows in the work area in a row at the lower end of the work area.

'Window' 'Close All'

With this command you can close all open windows in the work area.

'Window' 'Messages'

Shortcut: <Shift>+<Esc>

With this command you can open or close the message window with the messages from the last compiling, checking, or comparing procedure.

If the messages window is open, then a check (✓) will appear in front of the command.

'Window' 'Library Manager'

With this command you can open or close the library manager (see Chapter 4.2).

'Window' 'Log'

With this command you can open or close the Log window, where protocols of the online sessions can be displayed.

4.9 Help when you need it

An Online Help system is available to you for using **CoDeSys**. There you will find all the information that is also contained in this handbook.

'Help' 'Contents and Index'

With this command you can open the help topics window.

Under the **Contents** register card you will find the contents. The books can be opened and closed using a doubleclick or the corresponding button. Doubleclicking or activating the **Show** button on a highlighted topic will display the topic in the main window of help or in the index window.

Click on the **Index** register card to look for a specific word, and click on the **Search** register card to select a full-text search. Follow the instructions in the register cards.

Image 4.73: Help Topics Window

Main Help Window

In the main help window topics are displayed with index entries listed below them.

The following buttons available:

- **Help topics** opens the help topics window
- **Back** shows the help entry that was previously displayed
- **Print** opens the dialog box for printing
- << shows the help entry that comes prior in sequence to the present entry
- >> shows the help entry that is next in sequence

In addition you can use the following menu commands:

- With **'File' ''Print Topics'** you can print out the present help entry.
- If you use the **'Edit' "Copy"** command, the selected text will be copied into the clipboard. From here you can insert the text into other applications and use it there.
- If you use the **'Edit' "Annotate"** command, a dialog box will be opened. There is an editing field on the left side of the dialog box in which you can enter an annotation to the help page.

On the right side there are buttons for **storing** the text, for canceling the program, for **deleting** the notation, for **copying** a highlighted text on the clipboard , and for **pasting** a text from the clipboard.

If you have made an annotation to a help entry, a small green paper clip will appear in the upper left-hand corner. By clicking the mouse on the paper clip, you can open the dialog box with the annotation that has been made.

- If you would like to mark a page from help, then you can set a bookmark. To do so, choose the **"Define" "Bookmark"** command. A dialog box will appear in which you can enter a new name (The name of the page can serve as a starter) or can delete an old bookmark. If bookmarks were defined, then these will be displayed in the **"Bookmark"** menu. By choosing these menu items, you can access the desired page.
- Under **"Options"**, you can define whether the help window always appears in the foreground or in the background or in the standard setting.
- With **"Display previous topics"** under **"Options"**, you are furnished with a selection window with the previously displayed help topics. Doubleclick the entry you wish to view.
- Under **"Options"**, you can select the desired **"Font" in small, normal**, or **large**.
- If **"Options" "Use System Color"** has been chosen, help will not be displayed in the colors that were set, but in the system colors instead.

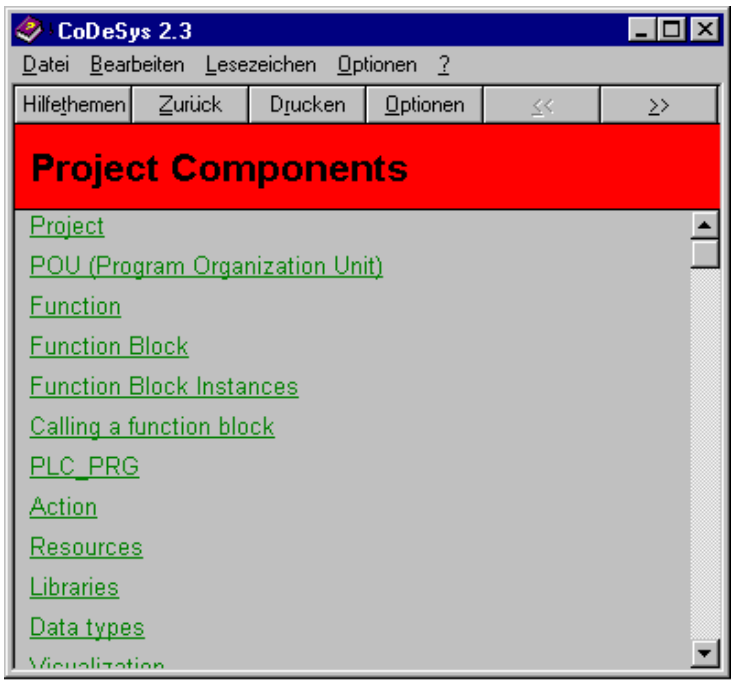


Image 4.74: Main Help Window

Index Window

The index window contains explanations about the menu commands, terms, or sequences.

The index window will always remain on the surface by default, unless the help option is placed in the background in the main window of help.

The following buttons are available:

- **Help topics** opens the help topics window
- **Back** shows the help entry that was previously displayed
- **Print** opens the dialog box for printing
- << shows the help entry directly prior to the present entry
- >> shows the help entry that is next in sequence

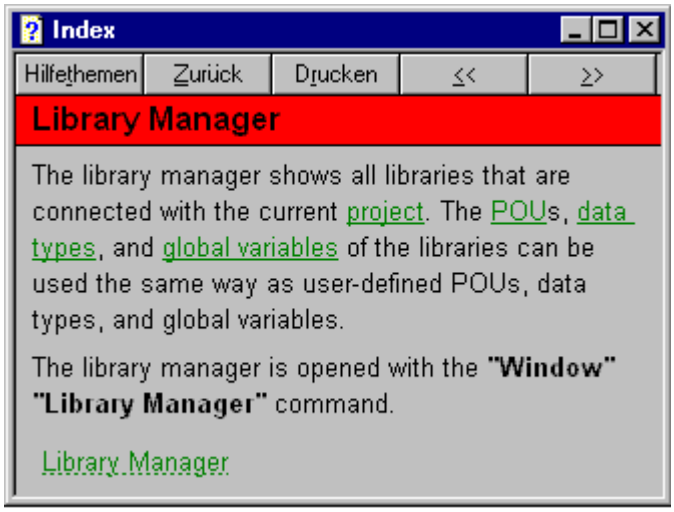

Index
Hilfethemen
Zurück
Drucken
<<
>>
Library Manager
The library manager shows all libraries that are connected with the current project. The POUs, data types, and global variables of the libraries can be used the same way as user-defined POUs, data types, and global variables.
The library manager is opened with the "Window" "Library Manager" command.
Library Manager

Image 4.75: Index Window

Context Sensitive Help

Shortcut: <F1>

You can use the <F1> key in an active window, in a dialog box, or above a menu command. When you perform a command from the menu, the help for the command called up at that time is displayed.

If you click on a warning or error message in the message window, the corresponding help text will appear.

You can also highlight a text (for example, a key word or a standard function) and have the help displayed for that item.

5 Editors in CoDeSys

All editors for POUs (Program Organization Units) consist of a declaration part and a body. These are separated by a screen divider that can be dragged as required by clicking it with the mouse and moving it up or down. The body can consist of other a text or a graphic editor; the declaration portion is always a text editor.

Print margins

The vertical and horizontal margins that apply when the editor contents are printed, are shown by red dashed lines if the **'Show print range'** option in the project options in the dialog **'Workspace'** was selected. The properties of the printer that was entered apply, as well as the size of the print layout selected in the **'File' 'Printer Setup'** menu. If no printer setup or no print layout is entered, a default configuration is used (Default.DFR and default printer). The horizontal margins are drawn as if the options 'New page for each object' or 'New page for each sub-object' were selected in 'Documentation settings'. The lowest margin is not displayed.

Note: An exact display of the print margins is only possible when a zoom factor of 100% is selected.

Comment

User comments must be enclosed in the special symbol sequences „(*“ and „*)“. Example: (*This is a comment.*)

Comments are allowed in all text editors, at any location desired, that is in all declarations, the IL and ST languages and in self-defined data types. If the Project is printed out using a **template**, the comment that was entered during variable declaration appears in text-based program components after each variable.

In the FBD and LD graphic editors, comments can be entered for each network. To do this, search for the network on which you wish to comment and activate **'Insert' 'Comment'**. In the Ladder Editor additionally a comment for each particular contact and coil can be added, if the corresponding options are activated in the menu 'Extras' 'Options'. In CFC there are special comment POUs which can be placed at will.

In SFC, you can enter comments about a step in the dialog for editing step attributes.

Nested comments are also allowed if the appropriate option in the **'Project' 'Options' 'Build Options'** dialog is activated.

In Online mode, if you rest the mouse cursor for a short time on a variable, the type and if applicable the address and comment of that variable are displayed in a tooltip.

Zoom to POU

Shortcut: <Alt>+<Enter>

With this command a selected POU is loaded into its editor. The command is available in the context menu (<F2>) or in the 'Extras' menu, if the cursor is positioned on the name of a POU in a text editor or if the POU box is selected in a graphic editor.

If you are dealing with a POU from a library, then the library manager is called up, and the corresponding POU is displayed.

Open instance

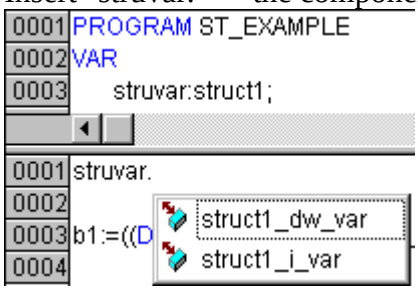
This command corresponds to the command 'Project' 'Open instance'. It is available in the context menu (<F2>) or in the 'Extras' menu, if the cursor is positioned on the name of a function block in a text editor or if the function block box is selected in a graphic editor.

Intellisense Function

If the option 'List components' is activated in the project options dialog for category 'Editor', then the "Intellisense" functionality will be available in all editors, in the Watch- and Receiptmanager, in the Visualization and in the Sampling Trace:

- If you insert a dot "." instead of an identifier, a selection box will appear, listing all local and global variables of the project. You can choose one of these elements and press 'Return' to insert it behind the dot. You can also insert the element by a doubleclick on the list entry.
- If you enter a function block instance or a structure variable followed by a dot, then a selection box listing all input and output variables of the corresponding function block resp. listing the structure components will appear, where you can choose the desired element and enter it by pressing 'Return' or by a doubleclick.

Example:
Insert "struvar." -> the components of structure struct1 will be offered:



5.1 Declaration Editor

The declaration editor is used to declare variables of POUs and global variables, for data type declarations, and in the Watch and Receipt Manager. It gives access to the usual Windows functions, and even those of the IntelliMouse can be used if the corresponding driver is installed.

In Overwrite mode, **'OV'** is shown in black on the status bar; switching between Overwrite and Insert modes can be accomplished with the <Ins> key.

The declaration of variables is supported by syntax coloring.

The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

Declaration Part

All variables to be used only in this POU are declared in the declaration part of the POU. These can include: input variables, output variables, input/output variables,

local variables, retain variables, and constants. The declaration syntax is based on the IEC61131-3 standard. An example of a correct declaration of variables in **CoDeSys**-Editor:

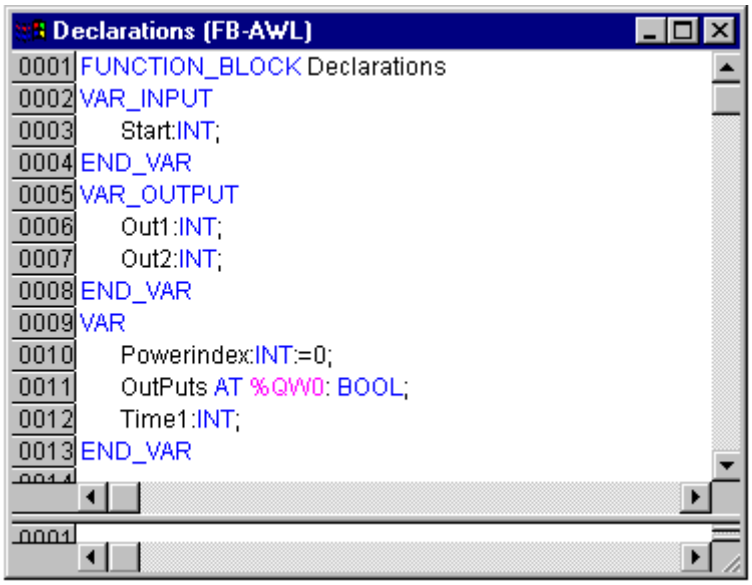


Image 5.1: Declaration Editor

Input Variable

Between the key words **VAR_INPUT** and **END_VAR**, all variables are declared that serve as input variables for a POU. That means that at the call position, the value of the variables can be given along with a call.

Example:

```
VAR_INPUT
   in1:INT       (* 1. Inputvariable*)
END_VAR
```

Output Variable

Between the key words **VAR_OUTPUT** and **END_VAR**, all variables are declared that serve as output variables of a POU. That means that these values are carried back to the POU making the call. There they can be answered and used further.

Example:

```
VAR_OUTPUT
   out1:INT;     (* 1. Outputvariable*)
END_VAR
```

Input and Output Variables

Between the key words **VAR_IN_OUT** and **END_VAR**, all variables are declared that serve as input and output variables for a POU.

Attention: With this variable, the value of the transferred variable is changed ("transferred as a pointer"). That means that the input value for such variables cannot be a constant.

Attention: With this variable, the value of the transferred variable is changed ("transferred as a pointer", Call-by-Reference). That means that the input value for

such variables cannot be a constant. For this reason, even the VAR_IN_OUT variables of a function block can not be read or written directly from outside via <functionblockinstance><in/outputvariable>.

Example:

```
VAR_IN_OUT
   inout1:INT;   (* 1. Inputoutputvariable *)
END_VAR
```

Local Variables

Between the keywords **VAR** and **END_VAR**, all of the local variables of a POU are declared. These have no external connection; in other words, they can not be written from the outside.

Example:

```
VAR
   loc1:INT;      (* 1. Local Variable*)
END_VAR
```

Remanent variables

Remanent variables can retain their value throughout the usual program run period. These include Retain variables and Persistent variables.

- Retain variables are identified by the keyword **RETAIN**. These variables maintain their value even after an uncontrolled shutdown of the controller as well as after a normal switching off and on of the controller (resp. at the command 'Online' 'Reset'. When the program is run again, the stored values will be processed further. A concrete example would be an piece-counter in a production line, that recommences counting after a power failure. Retain-Variables are reinitialized at a new download of the program unlike persistent variables.
  All other variables are newly initialized, either with their initialized values or with the standard initializations.
- Persistent variables are identified by the keyword **PERSISTENT**. Unlike Retain variables, these variables retain their value after a re-Download (resp. at the command 'Online' 'Reset Original'), but not at switching off and on the controller (i.e. not at the command 'Online' 'Reset'), because they are not saved in the "retain area". If persistent variables as well should maintain their values after a uncontrolled shutdown of the controller, then they have to be declared additionally as VAR RETAIN variables. A concrete example of "persistent Retain-Variables" would be a operations timer that recommences timing after a power failure.

Example:

```
VAR RETAIN
   rem1:INT;     (* 1. Retain variable*)
END_VAR
```

Attention:

- If a local variable is declared as VAR RETAIN, then exactly that variable will be saved in the "retain area" (like a global retain variable)

- If a local variable in a function block is declared as VAR RETAIN, then the complete instance of the function block will be saved in the "retain area" (all data of the POU), whereby only the declared retain variable will be handled as a retain.
- If a local variable in a function is declared as VAR RETAIN or VAR_PERSISTENT, this will be without any effect. If a local variable is declared as PERSISTENT in a function, then this will be without any effect also !

Constants, Typed Literals

Constants are identified by the key word **CONSTANT**. They can be declared locally or globally.

Syntax:

VAR CONSTANT
<Identifier>**:**<Type> **:=** <initialization>;
END_VAR

Example:

```
VAR CONSTANT
   con1:INT:=12;        (* 1. Constant*)
END_VAR
```

A listing of possible constants can be found in the Appendix F(Operanden in **CoDeSys**). See there also regarding the possibility of using typed constants (Typed Literals).

External variables

Global variables which are to be imported into the POU are designated with the keyword EXTERNAL. They also appear in the Watch window of the declaration part in Online mode.

If the VAR_EXTERNAL declaration does not match the global declaration in every respect, the following error message appears: "Declaration of '<var>' does not match global declaration!"

If the global variable does not exist, the following error message appears: "Unkown global variable: '<var>'!"

Example:

```
VAR EXTERNAL
   var_ext1:INT:=12;    (* 1st external variable *)
END_VAR
```

Keywords

Keywords are to be written in uppercase letters in all editors. Keywords may not be used as variables.

Variables declaration

A variables declaration has the following syntax:

<Identifier> {**AT**<Address>}**:**<Type> {**:=**<initialization>};

The parts in the braces {} are optional.

Regarding the identifier, that is the name of a variable, it should be noted that it may not contain spaces or umlaut characters, it may not be declared in duplicate and may not be identical to any keyword. Upper/lowercase writing of variables is ignored, in other words VAR1, Var1 and var1 are not different variables. Underlines in identifiers are meaningful, e.g. A_BCD and AB_CD are interpreted as different identifiers. Multiple consecutive underlines at the beginning of an identifier or within a identifier are not allowed. The length of the identifier, as well as the meaningful part of it, are unlimited.

All declarations of variables and data type elements can include initialization. They are brought about by the ":=" operator. For variables of elementary types, these initializations are constants. The default-initialization is 0 for all declarations.

Example:

```
var1:INT:=12;   (* Integer variable with initial value of 12*)
```

If you wish to link a variable directly to a definite address, then you must declare the variable with the keyword **AT**.

For faster input of the declarations, use the shortcut mode.

In function blocks you can also specify variables with incomplete address statements. In order for such a variable to be used in a local instance, there must be an entry for it in the variable configuration.

Pay attention to the possibility of an automatic declaration

Identifier

An identifier is a sequence of letters, numbers, and underscores that begins with a letter or an underscore.

The variable identifier may not contain any blank spaces or special characters, may not be declared more than once and cannot be the same as any of the keywords. Capitalization is not recognized which means that VAR1, Var1, and var1 are all the same variable. The underscore character is recognized in identifiers (e.g., "A_BCD" and "AB_CD" are considered two different identifiers). An identifier may not have more than one underscore character in a row. The first 32 characters are significant.

AT Declaration

If you wish to link a variable directly to a definite address, then you must declare the variable with the keyword **AT**. The advantage of such a procedure is that you can assign a meaningful name to an address, and that any necessary changes of an incoming or outgoing signal will only have to be made in one place (e.g., in the declaration).

Notice that variables requiring an input cannot be accessed by writing. A further restriction is that AT declarations can only be made for local and global variables, and not for input- and output variables from POUs.

Examples:

```
counter_heat7 AT %QX0.0: BOOL;
lightcabinetimpulse AT %IX7.2: BOOL;
download AT %MX2.2: BOOL;
```

If boolean variables are assigned to a Byte, Word or DWORD address, they occupy one byte with TRUE or FALSE, not just the first bit after the offset!

'Insert' 'Declarations keywords'

You can use this command to open a list of all the keywords that can be used in the declaration part of a POU. After a keyword has been chosen and the choice has been confirmed, the word will be inserted at the present cursor position.

You also receive the list, when you open the Input Assistant (<F2>) and choose the **Declarations** category.

'Insert' 'Type'

With this command you will receive a selection of the possible types for a declaration of variables. You also receive the list when you access the Input Assistant (<F2>).

The types are divided into these categories:

- Standard types BOOL, BYTE, etc.
- Defined types Structures, enumeration types, etc.
- Standard function blocks for instance declarations
- Defined function blocks for instance declarations

CoDeSys supports all standard types of IEC1131-3:

Examples for the use of the various types are found in the Appendix F.

Syntax Coloring

In all editors you receive visual support in the implementation and declaration of variables. Errors are avoided, or discovered more quickly, because the text is displayed in color.

A comment left unclosed, thus annotating instructions, will be noticed immediately; keywords will not be accidentally misspelled, etc.

The following color highlighting will be used:

Blue	Keywords
Green	Comments in the text editors
Pink	Special constants (e.g. TRUE/FALSE, T#3s, %IX0.0)
Red	Input error (for example, invalid time constant, keyword, written in lower case,...)
Black	Variables, constants, assignment operators, ...

Shortcut Mode

The declaration editor for **CoDeSys** allows you to use the shortcut mode. This mode is activated when you end a line with <Ctrl><Enter>

The following shortcuts are supported:

All identifiers up to the last identifier of a line will become declaration variable identifiers

The type of declaration is determined by the last identifier of the line. In this context, the following will apply:

B or BOOL	gives the result	BOOL
I or INT	gives the result	INT
R or REAL	gives the result	REAL
S or string	gives the result	STRING

If no type has been established through these rules, then the type is BOOL and the last identifier will not be used as a type (Example 1.).

Every constant, depending on the type of declaration, will turn into an initialization or a string (Examples 2. and 3.).

An address (as in %MD12) is extended around the AT... attribute(Example 4.).

A text after a semicolon (;) becomes a comment (Example 4.).

All other characters in the line are ignored (e.g., the exclamation point in Example 5.).

Examples:

Shortcut	Declaration
A	A: BOOL;
A B I 2	A, B: INT := 2;
ST S 2; A string	ST:STRING(2); (* A string *)
X %MD12 R 5 Real Number	X AT %MD12: REAL := 5.0;(* Real Number *)
B !	B: BOOL;

Autodeclaration

If the Autodeclaration option has been chosen in the Editor category of the Options dialog box , then a dialog box will appear in all editors after the input of a variable that has not yet been declared. With the help of this dialog box, the variable can now be declared.

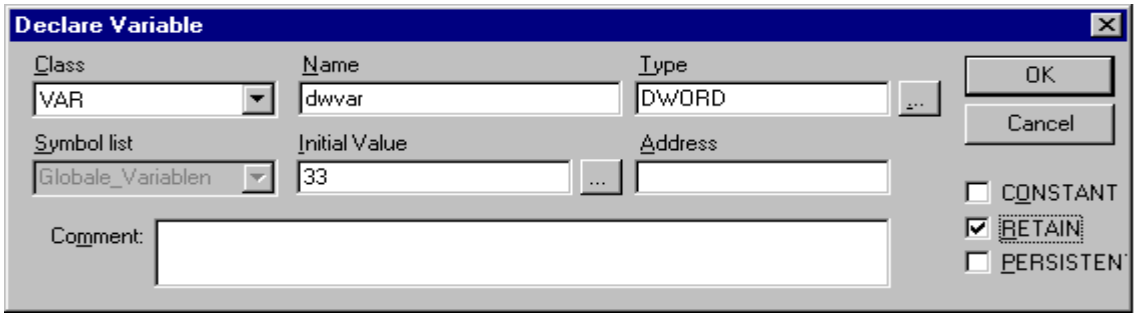


Image 5.2: Dialog Box for Declaration of Variables

With the help of the **Class** combobox, select whether you are dealing with a local variable (**VAR**), input variable((**VAR_INPUT**), output variable (**VAR_OUTPUT**), input/output variable (**VAR_INOUT**), or a global variable (**VAR_GLOBAL**).

With the **CONSTANT, RETAIN, PERSISTENT** options, you can define whether you are dealing with a constant or a remanent variable

The variable name you entered in the editor has been entered in the **Name** field, BOOL has been placed in the **Type** field. The ... button opens the Input Assistant dialog which allows you to select from all possible data types.

If ARRAY is chosen as the variable type, the dialog for entering array boundaries appears.

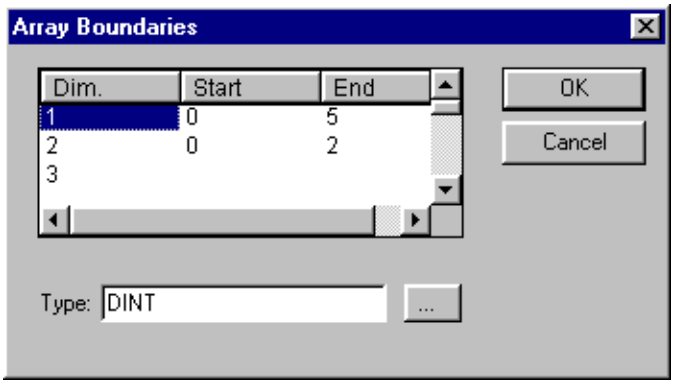


Image 5.3: Dialog for determining array boundaries during automatic declaration

For each of the three possible dimensions (**Dim.**), array boundaries can be entered under **Start** and **End** by clicking with the mouse on the corresponding field to open an editing space. The array data type is entered in the **Type** field. In doing this, the ... button can be used to call up an input assistant dialog.

Upon leaving the array boundaries dialog via the **OK** button, variable declarations in IEC format are set up based on the entries in the Type field in the dialog. Example: ARRAY [1..5, 1..3] OF INT

In the field **Initial value**, you may enter the initial value of the variable being declared. If this is an array or a valid structure, you can open a special initialization dialog via the ... button or <F2>, or open the input assistant dialog for other variable types.

In the initialization dialog for an array you are presented a list of array elements; a mouse click on the space following „:=“opens an editing field for entering the initial value of an element.

In the initialization dialog for a structure, individual components are displayed in a tree structure. The type and default initial value appear in brackets after the variable name; each is followed by „:=“. A mouse click on the field following „:=“ opens an editing field in which you can enter the desired initial value. If the component is an array, then the display of individual fields in the array can be expanded by a mouse click on the plus sign before the array name and the fields can be edited with initial values.

After leaving the initialization dialog with **OK**, the initialization of the array or the structure appears in the field **Initial value** of the declaration dialog in IEC format.

Example: x:=5,field:=2,3,struct2:=(a:=2,b:=3)

In the **Address** field, you can bind the variable being declared to an IEC address (AT declaration).

If applicable, enter a **Comment**. The comment can be formatted with line breaks by using the key combination <Ctrl> + <Enter>.

By pressing **OK**, the declaration dialog is closed and the variable is entered in the corresponding declaration editor in accordance to the IEC syntax.

Note: The dialog box for variable declaration you also get by the command 'Edit' 'Declare Variable' (see Chapter 2.5 General Editing Functions). If the cursor is resting on a variable in Online mode, the Autodeclare window can be opened with <Shift><F2> with the current variable-related settings displayed.

Line Numbers in the Declaration Editor

In offline mode, a simple click on a special line number will mark the entire text line.

In the online mode, a single click on a specific line number will open up or close the variable in this line, in case a structural variable is involved.

Declarations as table

If the **Declarations as table** option is activated in the Options dialog box in the category, the declaration editor looks like a table. As in a card-index box, you can select the register cards of the respective variable types and edit the variables.

For each variable you are given the following entry fields.

Name:	Input the identifier of the variable.
Address:	If necessary, input the address of the variable (AT declaration)
Type:	Input the type of the variable. (Input the function block when instantiating a function block)
Initial:	Enter a possible initialization of the variable (corresponding to the ":= " assignment operator).
Comment:	Enter a comment here.

Both of the display types of the declaration editor can be changed without causing any problems. In the online mode, there are no differences for the display.

In order to edit a new variable, select the **'Insert' 'New Declaration'** command.

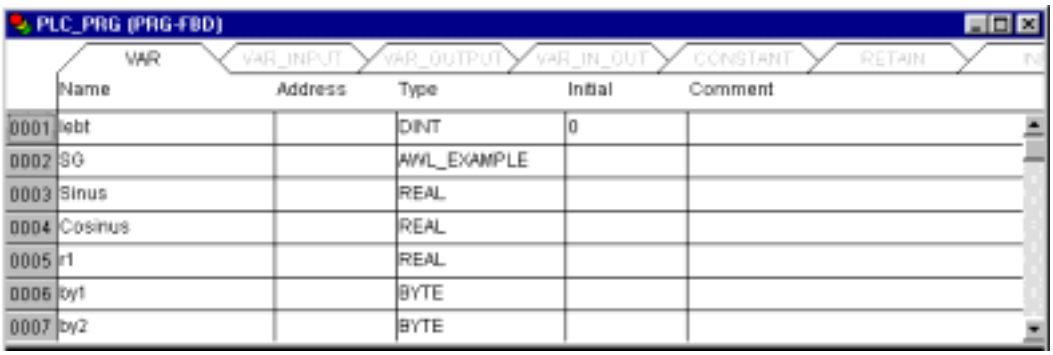


Image 5.4: Declaration Editor as a Table

'Insert' 'New Declaration'

With this command you bring a new variable into the declaration table of the declaration editor. If the present cursor position is located in an field of the table, then the new variable will be pasted in the preceding line; otherwise, the new variable is pasted at the end of the table. Moreover, you can paste a new declaration at the end of the table by using the right arrow key or the tab key in the last field of the table.

You will receive a variable that has "Name" located in the **Name** field, and "Bool" located in the **Type** field, as its default setting. You should change these values to the desired values. Name and type are all that is necessary for a complete declaration of variables.

Pragma command

The pragma instruction is used to affect the properties of a variable concerning the compilation process. It can be used in with supplementary text in a program line of the declaration editor or in its own line.

The pragma instruction is enclosed in curly brackets, upper- and lower-case are ignored: { <Instruction text> }

If the compiler cannot meaningfully interpret the instruction text, the entire pragma is handled as a comment and read over. A warning is issued, however: „Ignore compiler directive ‚<Instruction text>'!“

Depending on the type and contents of pragma, the pragma either operates on the line in which it is located or on all subsequent lines until it is ended by an appropriate pragma, or the same pragma is executed with different parameters, or the end of the file is reached. By file we mean here: declaration part, implementation portion, global variable list, type declaration.

The opening bracket may immediately follow a variable name. Opening and closing brackets must be located on the same line.

The following pragma may currently be used:

{flag [<flags>] [off|on]}

This pragma allows the properties of a variable declaration to be affected.

<flags> can be a combination of the following flags:

noinit The variable will not be initialized.

nowatch	The variable can not be monitored.
noread	The variable is exported to the symbol file without read permission.
nowrite	The variable is exported to the symbol file without write permission.
noread, nowrite	The variable is not exported to the symbol file

With the „**on**“ modifier, the pragma operates on all subsequent variable declarations until it is ended by the pragma {flag **off**}, or until overwritten by another {flag <flags> on} pragma.

Without the „on“ or „off“ modifier, the pragma operates only on the current variable declaration (that is the declaration that is closed by the next semicolon).

Examples: The variable a will not be initialized and will not be monitored. The variable b will not be initialized:

```
VAR
        a : INT  {flag noinit, nowatch};
        b : INT  {flag noinit };
END_VAR
```

```
VAR
        {flag noinit, nowatch on}
        a : INT;
        {flag noinit on}
        b : INT;
        {flag off}
END_VAR
```

Neither variable will be initialized:

```
{flag noinit on}
VAR
        a : INT;
        b : INT;
END_VAR
{flag off}
```

```
VAR
        {flag noinit on}
        a : INT;
        b : INT;
        {flag off}
END_VAR
```

The flags „**noread**“ and „**nowrite**“ are used in a POU that has read and/or write permission to provide selected variables with restricted access rights. The default for the variable is the same as the setting for the POU in which the variable is declared. If a variable has neither read nor write permission, it will not be exported into the symbol file.

Examples: If the POU has read and write permission, then with the following pragmas variable a can only be exported with write permission, while variable b can not be exported at all:

```
VAR
        a : INT  {flag noread};
        b : INT {flag noread, nowrite};
END_VAR
```

```
VAR
        { flag noread on}
        a : INT;
        { flag noread, nowrite on}
        b : INT;
        {flag off}
END_VAR
```

Neither variable a nor b will be exported to the symbol file:

```
{ flag noread, nowrite on }
VAR
        a : INT;
        b : INT;
END_VAR
{flag off}
```

```
VAR
        { flag noread, nowrite on }
        a : INT;
        b : INT;
        {flag off}
END_VAR
```

The pragma operates **additively** on all subsequent variable declarations.

Example: all POUs in use will be exported with read and write permission

```
a : afb;
...
FUNCTION_BLOCK afB
    VAR
      b : bfb {flag nowrite};
      c : INT;
    END_VAR
...
FUNCTION_BLOCK bfB
    VAR
      d : INT {flag noread};
      e : INT {flag nowrite};
    END_VAR
```

„a.b.d“: Will not be exported

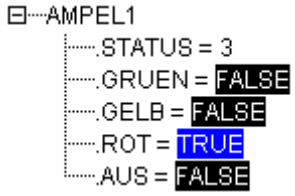
„a.b.e“: Will be exported only with read permission

„a.c“: Will be exported with read and write permission.

Declaration Editors in Online Mode

In online mode , the declaration editor changes into a monitor window. In each line there is a variable followed by the equal sign (=) and the value of the variable. If the variable at this point is undefined, three question marks (???) will appear. For function blocks, values are displayed only for open instances (command: 'Project' 'Open instance').

In front of every multi-element variable there is a plus sign. By pressing <Enter> or after doubleclicking on such a variable, the variable is opened up. In the example, the traffic signal structure would be opened up.



When a variable is open, all of its components are listed after it. A minus sign appears in front of the variable. If you doubleclick again or press <Enter>, the variable will be closed, and the plus sign will reappear.

Pressing <Enter> or doubleclicking on a single-element variable will open the dialog box to write a variable (see Chapter 2.6, ' General Online Functions'). Here it is possible to change the present value of the variable. In the case of Boolean variables, no dialog box appears; these variables are toggled.

The new value is displayed after the variable, in pointed brackets and in turquoise color, and remains unchanged. If the **'Online' 'Write values'** command is given, then all variables are placed in the selected list and are once again displayed in black.

If the **'Online' 'Force values'** command is given, then all variables will be set to the selected values, until the 'Release force' command is given. In this event, the color of the force value changes to red

5.2 The Text Editors

The text editors used for the implementation portion (the Instruction List editor and the Structured Text editor) of **CoDeSys** provide the usual Windows text editor functions.

The implementation in the text editors is supported by syntax coloring.

In Overwrite mode the status bar shows a black **OV**. You can switch between Overwrite mode and Insert mode by key <Ins>.

The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

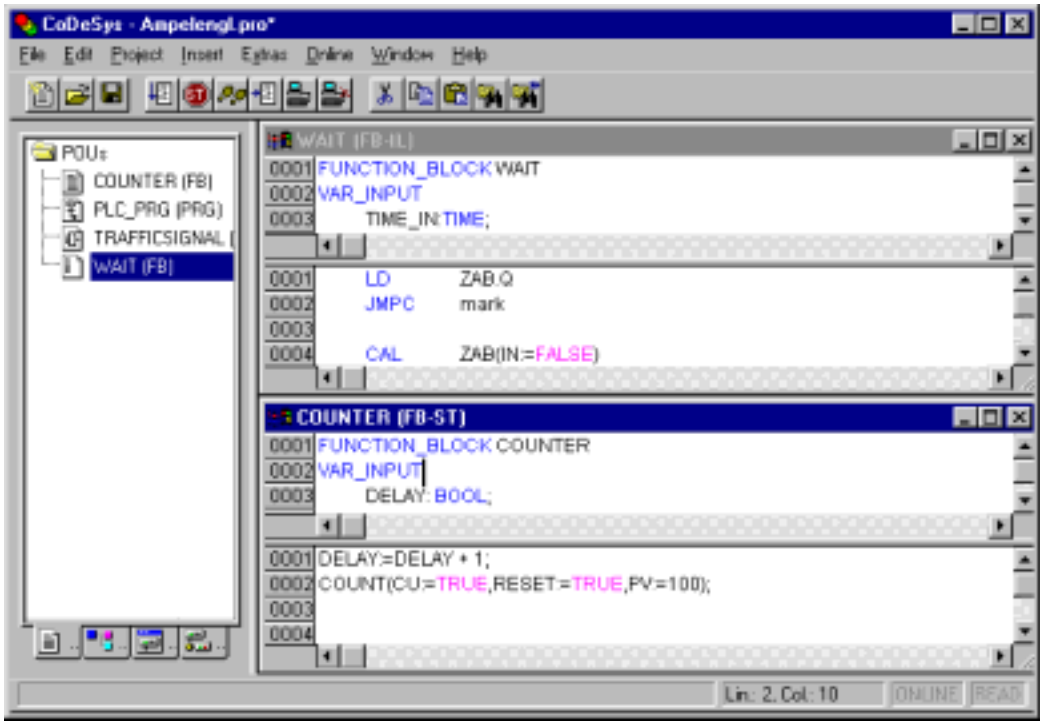


Image 5.5: Text Editors for the Instruction List and Structured Text

The text editors use the following menu commands in special ways:

'Insert''Operators'

With this command all of the operators available in the current language are displayed in a dialog box.

If one of the operators is selected and the list is closed with **OK**, then the highlighted operator will be inserted at the present cursor position. (This is managed here just as it is in the Input Assistant).

'Insert' 'Operand'

With this command all variables in a dialog box are displayed. You can select whether you would like to display a list of the global, the local, or the system variables.

If one of the operands is chosen, and the dialog box is closed with **OK**, then the highlighted operand will be inserted at the present cursor position. (This is managed here just as it is in the Input Assistant).

'Insert' 'Function'

With this command all functions will be displayed in a dialog box. You can choose whether to have a list displaying user-defined or standard functions.

If one of the functions is selected and the dialog box is closed with **OK**, then the highlighted function will be inserted at the current cursor position. (The management will proceed, as in the input selection.)

If the **With arguments** option was selected in the dialog box, then the necessary input and output variables will also be inserted.

'Insert' 'Function Block'

With this command all function blocks are displayed in a dialog box. You can choose whether to have a list displaying user-defined or standard function blocks.

If one of the function blocks is selected and the dialog box is closed with **OK**, then the highlighted function block will be inserted at the current cursor position. (This is managed here just as it is in the Input Assistant).

If the **With arguments** option was selected in the dialog box, then the necessary input variables of the function block will also be inserted.

Calling POUs with output parameters

The output parameters of a called POU can be directly assigned upon being called in the text languages IL and ST. Example: Output parameter out1 of afbinst is assigned variable a.

IL: CAL afbinst(in1:=1, out1=>a)

ST: afbinst(in1:=1, out1=>a);

The text editors in Online mode

The online functions in the editors are set breakpoint and single step processing (steps). Together with the monitoring, the user thus has the debugging capability of a modern Windows standard language debugger.

In Online mode, the text editor window is vertically divided in halves. On the left side of the window you will then find the normal program text; on the right side you will see a display of the variables whose values were changed in the respective lines.

The display is the same as in the declaration part. That means that when the PLC is running, the present values of the respective variables will be displayed.

The following should be noted when monitoring expressions or Bit-addressed variables: in the case of expressions, the value of the entire expression is always displayed. Example: a AND b is displayed in blue or with „:=TRUE“ if both a and b are TRUE. For Bit-addressed variables, the bit value that is addressed is always monitored (e.g. a.3 is displayed in blue or with „:=TRUE, if a has the value 4).

If you place the mouse pointer briefly above a variable, then the type, the address and the comment about the variable will be displayed in a Tooltip.

'Extras' 'Monitoring Options'

With this command you can configure your monitoring window. In the text editors, the window is divided into two halves during monitoring. The program is located in the left half. In the right half, all variables that are located in the corresponding program line are monitored.

You can specify the Monitor Window **Width** and which **Distance** two variables should have in a line. An distance declaration of 1 corresponds, in this case, to a line height in the selected font.

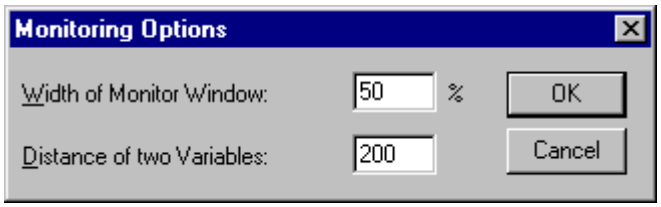


Image 5.6: Monitoring Options Dialog Box

Breakpoint PositionsSince in **CoDeSys** several IL lines are internally combined into a single C-code line, breakpoints can not be set in every line. Breakpoint positions include all positions in a program at which values of variables can change or where the program flow branches off. (Exception: function calls. If necessary, a breakpoint in the function must be set here.) At the positions lying inbetween, a breakpoint would not even make sense, since nothing has been able to change in the data since the preceding breakpoint position.

This results in the following breakpoint positions in the IL:

- At the start of the POU
- At every LD, LDN (or, in case a LD is located at a label, then at the label)
- At every JMP, JMPC, JMPCN
- At every label
- At every CAL, CALC, CALCN
- At every RET, RETC, RETCN
- At the end of the POU

Structured Text accommodates the following breakpoint positions:

- At every assignment
- At every RETURN and EXIT instruction
- in lines where conditions are being evaluated (WHILE, IF, REPEAT)
- At the end of the POU

Breakpoint positions are marked by the display of the line number field in the color which is set in the project options.

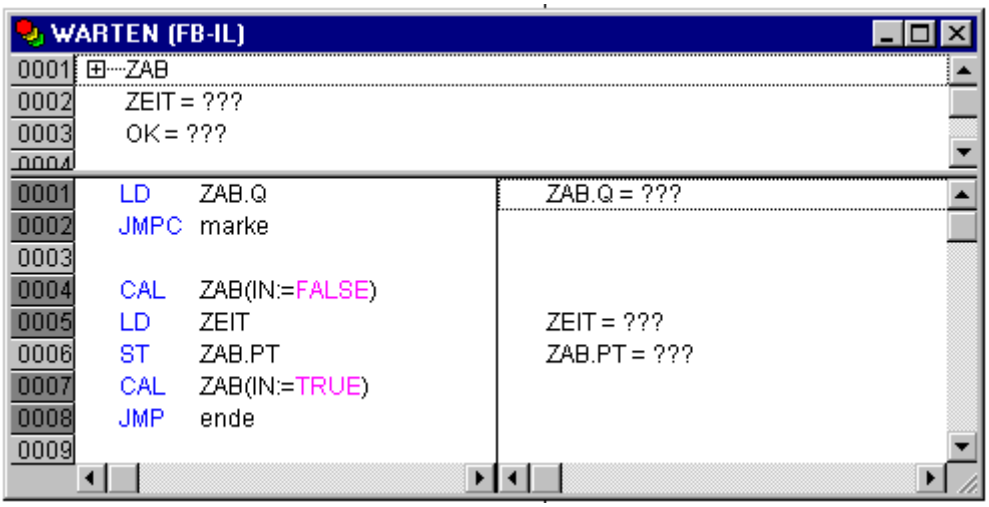


Image 5.7: IL Editor with Possible Breakpoint Positions (darker number fields)

How do you set a breakpoint?

In order to set a breakpoint, click the line number field of the line where you want to set a breakpoint. If the selected field is a breakpoint position, then the color of the line numbers field will change from dark gray to light blue, and the breakpoint will be activated in the PLC.

Deleting Breakpoints

Correspondingly, in order to delete a breakpoint, click on the line number field of the line with the breakpoint to be deleted.

Setting and deleting of breakpoints can also be selected via the menu ('**Online**' '**Toggle Breakpoint**'), via the function key <F9>, or via the symbol in the tool bar.

What happens at a breakpoint?

If a breakpoint is reached in the PLC, then the screen will display the break with the corresponding line. The line number field of the line where the PLC is positioned will appear in red. The user program is stopped in the PLC.

If the program is at a breakpoint, then the processing can be resumed with '**Online**' '**Run**'.

In addition, with '**Online**' '**Step over**' or '**Step in**' you can cause the program to run to the next breakpoint position. If the instruction where you are located is a CAL command, or, if there is a function call in the lines up to the next breakpoint position, then you can use '**Step over**' to bypass the function call. With '**Step in**', you will branch to the open POU

Line Number of the Text Editor

The line numbers of the text editor give the number of each text line of an implementation of a POU.

In Off-line mode, a simple click on a special line number will mark the entire text line.

In Online mode, the background color of the line number indicates the breakpoint status of every line. The standard settings for the colors are

- dark gray: This line is a possible position for a breakpoint.
- light blue: a breakpoint has been set in this line.
- red: The program has reached this point.

In Online mode, simply clicking the mouse will change the breakpoint status of this line.

5.2.1 The Instruction List Editor

This is how a POU written in the IL looks under the corresponding **CoDeSys** editor:

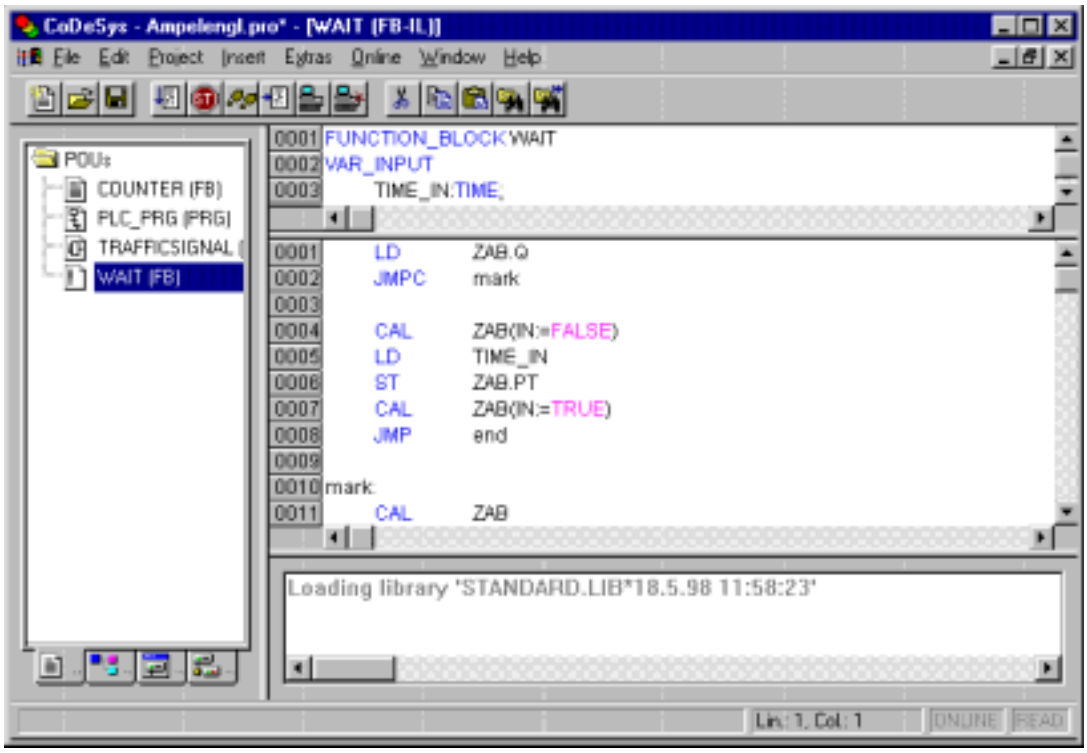


Image 5.8: IL Editor

All editors for POUs consist of a declaration part and a body. These are separated by a screen divider.

The Instruction List editor is a text editor with the usual capabilities of Windows text editors. The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).Multiline POU calls are also possible: Example:

```
CAL CTU_inst(
      CU:=%IX10,
        PV:=(
        LD   A
       ADD  5
        )
      )
```

For information concerning the language, see Chapter 2.2.1, Instruction List (IL).

IL in Online mode

With the **'Online' 'Flow control'** command, an additional field in which the accumulator contents is displayed is inserted in the IL editor on the left side of every line.

For further information concerning the IL editor in Online mode, see above 'The Text Editors in Online Mode'.

5.2.2 The Editor for Structured Text

This is how a POU written in ST appears under the corresponding **CoDeSys** editor:

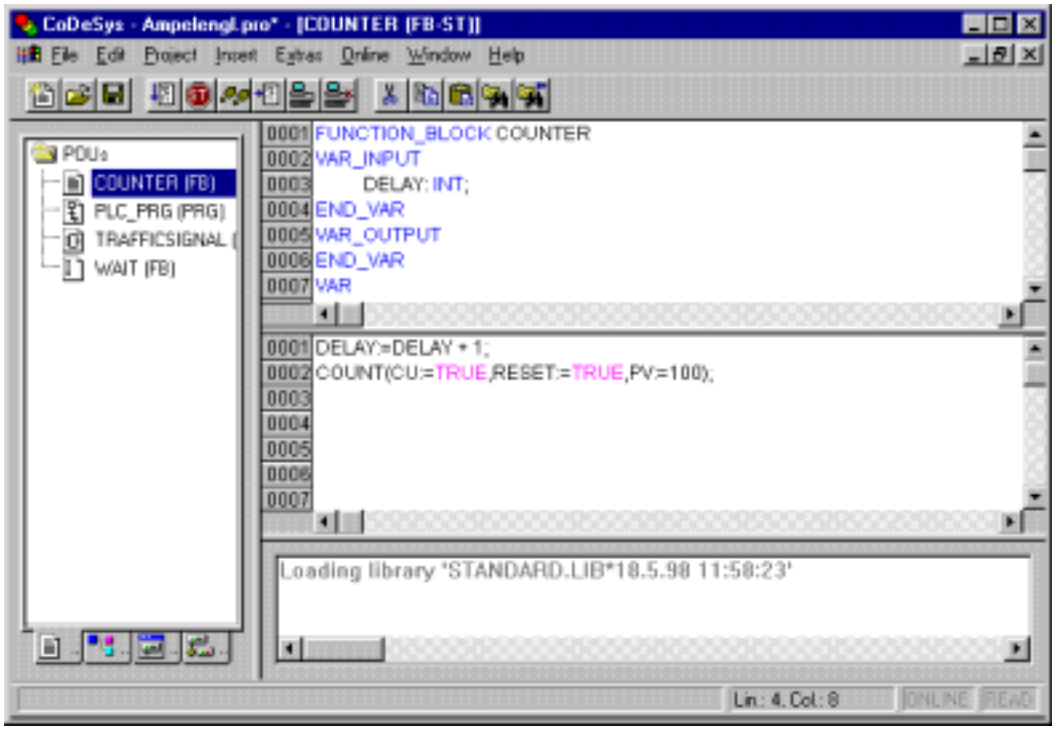


Image 5.9: Editor for Structured Text

All editors for POUs consist of a declaration part and a body. These are separated by a screen divider.

The editor for Structured Text is a text editor with the usual capabilities of Windows text editors. The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

For information about the ST editor in Online mode, read Text Editors in Online Mode.

For information about the ST editor in Online mode, read above 'The Text Editors in Online Mode'.

For information about the language, read the chapter 2.2.2, Structured Text (ST).

5.3 The Graphic Editors

The editors of the graphically oriented languages, sequential function chart SFC, ladder diagram LD and function block diagram FBD and of free graphic function block diagrams have many points in common. In the following paragraphs these features will be summarized; the specific descriptions of LD, FBD an CFC, as well as the Sequential Function Chart language SFC follow in separate sections. The implementation in the graphics editors is supported by syntax coloring.

Zoom

Objects such as POUs, actions, transitions etc. in the languages SFC, LD, FBD, CFC and in visualizations can be enlarged or reduced in size with a zoom function. All elements of the window contents of the implementation part are affected; the declaration part remains unchanged.

In standard form, every object is displayed with the zoom level 100%. The zoom level that is set is saved as an object property in the project.

The printing of project documentation always occurs at the 100% display level!

The zoom level can be set through a selection list in the toolbar. Values between 25% and 400% can be selected; individual values between 10% and 500% can be entered manually.

The selection of a zoom level is only available if the cursor rests on an object created in a graphical language or a visualization object.

Even with the object zoomed, cursor positions in the editors can be further selected and even reached with the arrow keys. Text size is governed by the zoom factor and the font size that is set.

The execution of all editor menu features (e.g. inserting a box) as a function of cursor position is available at all zoom levels, taking the same into account.

In Online mode, each object is displayed according to the zoom level that has been set; Online functionality is available without restriction.

When the IntelliMouse is used, an object can be enlarged/reduced by pressing the <CTRL> key and at the same time turning the wheel forward or backwards.

Network

In the LD and FBD editors, the program is arranged in a list of networks. Each network is designated on the left side by a serial network number and has a structure consisting of either a logical or an arithmetic expression, a program, function or function block call, and a jump or a return instruction.

Label

Each network has a label that can optionally be left empty. This label is edited by clicking the first line of the network, directly next to the network number. Now you can enter a label, followed by a colon.

Network Comments, Networks with Linebreaks, 'Extras' 'Options'

Every network can be supplied with a multi-lined comment. In the dialog 'Function Block and Ladder Diagram Options', which can be opened by executing the command '**Extras**' '**Options**', you can do the settings concerning comments and linebreaks: In the **maximum comment size** you can enter the maximum number of lines to be made available for a network comment (The default value here is 4.) In the field **minimum comment size y**ou can enter the number of lines that generally should be reserved for. If, for example, the number 2 is entered, then, at the start of each network there will be two empty lines after the label line. The default value here is 0, which has the advantage of allowing more networks to fit in the screen area.

If the minimal comment size is greater than 0, then in order to enter a comment you simply click in the comment line and then enter the comment. Otherwise you must next select the network to which a comment is to be entered, and use '**Insert**' '**Comment**' to insert a comment line. In contrast to the program text, comments are displayed in gray.

In the Ladder editor you can also assign an individual comment to each contact or coil. For this activate the option **Comments per Contact** and insert in the edit field **Lines for variable comment** the number of lines which should be reserved and displayed for the comment. If this setting is done, a comment field will be displayed in the editor above each contact and coil where you can insert text.

If the option **Comments per Contact** is activated, then in the Ladder editor also the number of lines (**Lines for variable text :**) can be defined which should be used for the variable name of the contact resp. coil. This is used to display even long names completely by breaking them into several lines.

In the Ladder editor it is possible to force linebreaks in the networks as soon as the network length would exceed the given window size and some of the elements would not be visible. For this activate the option **Networks with Linebreaks.**

'Insert' 'Network (after)' or 'Insert' 'Network (before)'

Shortcut: <Shift>+<T> (Network after)

In order to insert a new network in the FBD or the LD editor, select the '**Insert**' '**Network (after)**' or **the** '**Insert**' '**Network (before)**' command, depending on whether you want to insert the new network before or after the present network. The present network can be changed by clicking the network number. You will recognize it in the dotted rectangle under the number. With the <Shift key> and a mouse click you can select from the entire area of networks, from the present one to the one clicked.

The network editors in the online mode

In the FBD and the LD editors you can only set breakpoints for networks. The network number field of a network for which a breakpoint has been set, is displayed in blue. The processing then stops in front of the network, where the breakpoint is

located. In this case, the network number field is displayed in red. With single step processing (steps), you can jump from network to network.

All values are monitored upon entering and exiting network POUs (Program Organization Units).

The following should be noted when monitoring expressions or Bit-addressed variables: In expressions, e.g. a AND b, used as transition condition or function block input, the value of the whole expression is always displayed (a AND b is shown in blue or as :=TRUE, if a and b are TRUE). For Bit-addressed variables, the bit value that is addressed is always monitored (e.g. a.3 is displayed in blue or with „:=TRUE, if a has the value 4)

The flow control is run with the **'Online' 'Flow control'** command. Using the flow control, you can view the present values that are being carried in the networks over the connecting lines. If the connecting lines do not carry Boolean values, then the value will be displayed in a specially inserted field. The monitor fields for variables that are not used (e.g. in the function SEL) are displayed in a shade of grey. If the lines carry Boolean values, then they will be shaded blue, in the event that they carry TRUE. Therefore, you can accompany the flow of information while the PLC is running.

If you place the mouse pointer briefly above a variable, then the type, the address and the comment about the variable will be displayed in a Tooltip.

5.3.1 The Function Block Diagram Editor

This is how a POU written in the FBD under the corresponding **CoDeSys** editor looks:

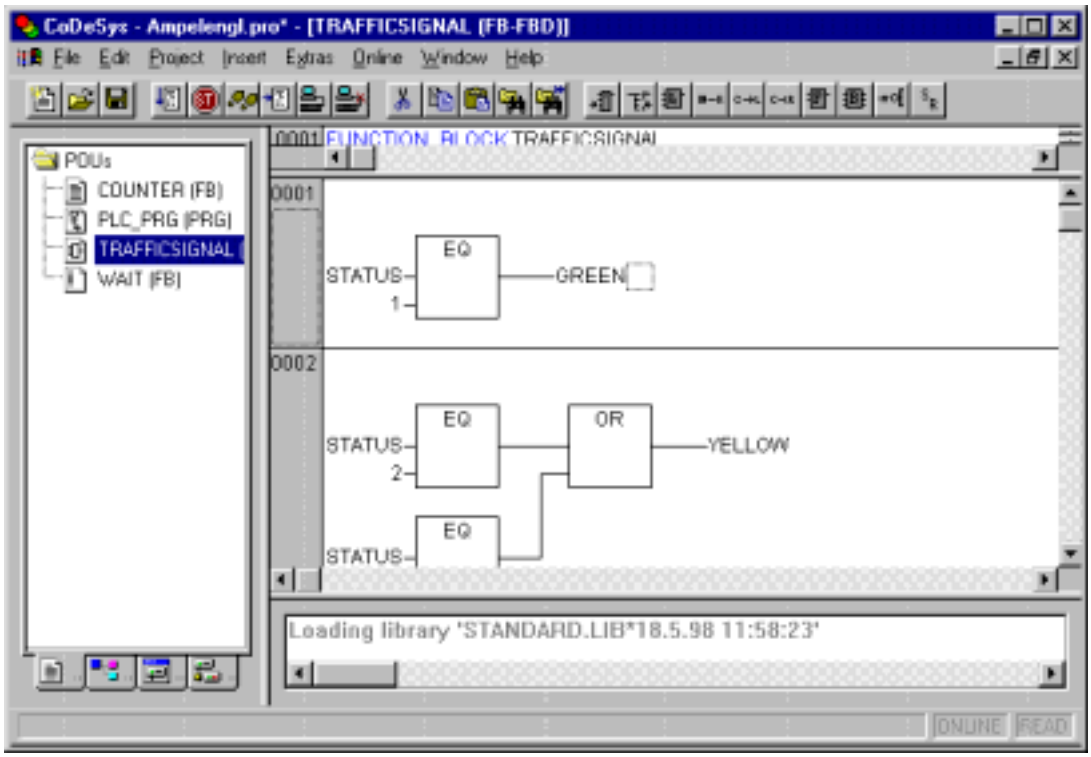


Image 5.10: Editor for the Function Block Diagram

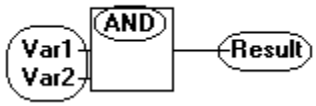
The Function Block Diagram editor is a graphic editor. It works with a list of networks, in which every network contains a structure that displays, respectively, a logical or an arithmetical expression, the calling up of a function block, a function, a program, a jump, or a return instruction.The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

Cursor positions in FBD

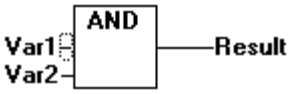
Every text is a possible cursor position. The selected text is on a blue background and can now be changed.

You can also recognize the present cursor position by a dotted rectangle. The following is a list of all possible cursor positions with an example:

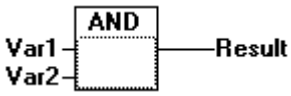
1) Every text field (possible cursor positions framed in black):



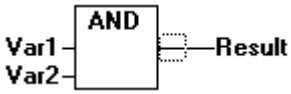
2) Every input:



3) Every operator, function, or function block:

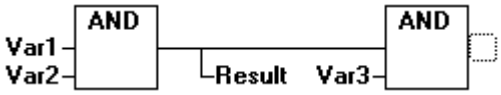


4) Outputs, if an assignment or a jump comes afterward:



5) The lined cross above an assignment, a jump, or a return instruction:

6) Behind the outermost object on the right of every network ("last cursor position," the same cursor position that was used to select a network):



7) The lined cross directly in front of an assignment:

How to set the cursor: The cursor can be set at a certain position by clicking the mouse, or with the help of the keyboard.

Using the arrow keys, you can jump to the nearest cursor position in the selected direction at any time. All cursor positions, including the text fields, can be accessed this way. If the last cursor position is selected, then the <up> or <down> arrow keys can be used to select the last cursor position of the previous or subsequent network.

An empty network contains only three question marks "???". By clicking behind these, the last cursor position is selected.

'Insert' 'Assign'

Symbol: **Shortcut: <Ctrl>+<A>**

This command inserts an assignment.

Depending on the selected position (see above 'Cursor positions in FBD'), insertion takes place directly in front of the selected input (Cursor Position 2), directly after the selected output (Cursor Position 4) or at the end of the network (Cursor Position 6).

For an inserted assignment, a selection can be made accompanying the entered text "???", and the assignment can be replaced by the variable that is to be assigned. For this you can also use the Input Assistant.

In order to insert an additional assignment to an existing assignment, use the **'Insert' 'Output'** command.

'Insert' 'Jump'

Symbol: **Shortcut: <Ctrl>+<L>**

This command inserts a jump.

Depending on the selected position (see above 'Cursor positions in FBD'), insertion takes place directly in front of the selected input (Cursor Position 2), directly after the selected output (Cursor Position 4) or at the end of the network (Cursor Position 6).

For an inserted jump, a selection can be made accompanying the entered text "???", and the jump can be replaced by the label to which it is to be assigned.

'Insert' 'Return'

Symbol: **Shortcut: <Ctrl>+<R>**

This command inserts a RETURN instruction.

Depending on the selected position (see above 'Cursor positions in FBD'), insertion takes place directly in front of the selected input (Cursor Position 2),directly after the selected output (Cursor Position 4), directly before the selected line cross (Cursor Position 5), or at the end of the network (Cursor Position 6)

'Insert' 'Box'

Symbol: **Shortcut: <Ctrl>+<B>**

With this command, operators, functions, function blocks and programs can be inserted. First of all, it is always inserted an “AND” operator. This can be converted by Selection and Overwrite of the type text („AND“) into every other operator, into

every function, into every function block and every program. You can select the desired POU by using Input Assistant (<F2>). If the new selected block has another minimum number of inputs, these will be attached. If the new block has a smaller highest number of inputs, the last inputs will be deleted.

In functions and function blocks, the formal names of the in- and outputs are displayed.

In function blocks there exists an editable instance field above the box. If another function block that is not known is called by changing the type text, an operator box with two inputs and the given type is displayed. If the instance field is selected, Input Assistant can be obtained via <F2> with the categories for variable selection.

The newest POU is inserted at the selected position (see above 'Cursor positions in FBD'):

- If an input is selected (Cursor Position 2), then the POU is inserted in front of this input. The first input of this POU is linked to the branch on the left of the selected input. The output of the new POU is linked to the selected input.
- If an output is selected (Cursor Position 4), then the POU is inserted after this output. The first input of the POU is connected with the selected output. The output of the new POU is linked to the branch with which the selected output was linked.
- If a POU, a function, or a function block is selected (Cursor Position 3), then the old element will be replaced by the new POU.
- As far as possible, the branches will be connected the same way as they were before the replacement. If the old element had more inputs than the new one, then the unattachable branches will be deleted. The same holds true for the outputs.
- If a jump or a return is selected, then the POU will be inserted before this jump or return. The first input of the POU is connected with the branch to the left of the selected element. The output of the POU is linked to the branch to the right of the selected element.
- If the last cursor position of a network is selected (Cursor Position 6), then the POU will be inserted following the last element. The first input of the POU is linked to the branch to the left of the selected position.

All POU inputs that could not be linked will receive the text "???". This text must be clicked and changed into the desired constant or variable.

If there is a branch to the right of an inserted POU, then the branch will be assigned to the first POU output. Otherwise the outputs remane unassigned.

‘Insert’ ‘Input'

Symbol: **Shortcut: <Ctrl>+<U>**

This command inserts an operator input. With many operators, the number of inputs may vary. (For example, ADD can have 2 or more inputs.)

In order to extend such an operator by an input, you need to select the input in front of which you wish to insert an additional input (Cursor Position 1); or you must select the operator itself (Cursor Position 3), if a lowest input is to be inserted (see above 'Cursor positions in FBD').

The inserted input is allocated with the text "???". This text must be clicked and changed into the desired constant or variable. For this you can also use the Input Assistant.

'Insert' 'Output'

Symbol:

This command inserts an additional assignment into an existing assignment. This capability serves the placement of so-called assignment combs; i.e., the assignment of the value presently located at the line to several variables.

If you select the lined cross above an assignment (Cursor Position 5) (see above 'Cursor positions in FBD') or the output directly in front of it (Cursor Position 4), then there will be another assignment inserted after the ones already there.

If the line cross directly in front of an assignment is selected (Cursor Position 4), then another assignment will be inserted in front of this one.

The inserted output is allocated with the text "???". This text must be clicked and changed into the desired variable. For this you can also use the Input Assistant.

'Extras' 'Negation'

Symbol: **Shortcut: <Ctrl>+<N>**

With this command you can negate the inputs, outputs, jumps, or RETURN instructions. The symbol for the negation is a small circle at a connection.

If an input is selected (Cursor Position 2) (see above 'Cursor positions in FBD'), then this input will be negated.

If an output is selected (Cursor Position 4), then this output will be negated.

If a jump or a return is marked, then the input of this jump or return will be negated.

A negation can be canceled through renewed negation.

'Extras' 'Set/Reset'

Symbol:

With this command you can define outputs as Set or Reset Outputs. A grid with Set Output is displayed with [S], and a grid with Reset Output is displayed with [R].

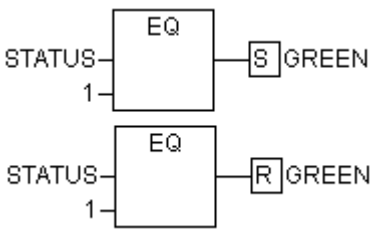


Image 5.11: Set/Reset Outputs in FBD

An *Output Set* is set to TRUE, if the grid belonging to it returns TRUE. The output now maintains this value, even if the grid jumps back to FALSE.

An *Output Reset* is set to FALSE, if the grid belonging to it returns FALSE. The output maintains its value, even if the grid jumps back to FALSE.

With multiple executions of the command, the output will alternate between set, reset, and normal output.

Cutting, Copying, Pasting, and Deleting in FBD

The commands used to **'Cut'**, **'Copy'**, **'Paste'**, and **'Delete'** are found under the **'Edit'** menu item.

If a line cross is selected (Cursor Position 5) (see above 'Cursor positions in FBD'), then the assignments, jumps, or RETURNS located below the crossed line will be cut, deleted, or copied.

If a POU is selected (Cursor Position 3), then the selected object itself, will be cut, deleted, or copied, along with all of the branches dependent on the inputs, with the exception of the first (highest position) branch.

Otherwise, the entire branch located in front of the cursor position will be cut, deleted, or copied.

After copying or cutting, the deleted or copied part is located on the clipboard and can now be pasted, as desired.

In order to do so, you must first select the pasting point. Valid pasting points include inputs and outputs.

If a POU has been loaded onto the clipboard (As a reminder: in this case all connected branches except the first are located together on the clipboard), the first input is connected with the branch before the pasting point.

Otherwise, the entire branch located in front of the pasting point will be replaced by the contents of the clipboard.

In each case, the last element pasted is connected to the branch located in front of the pasting point.

Note: The following problem is solved by cutting and pasting: A new operator is inserted in the middle of a network. The branch located on the right of the operator is now connected with the first input, but should be connected with the second input. You can now select the first input and perform the command **'Edit' 'Cut'**. Following this, you can select the second input and perform the command **'Edit' 'Paste'**. This way, the branch is dependent on the second input.

The Function Block Diagram in the Online Mode

In the Function Block Diagram, breakpoints can only be set to networks. If a breakpoint has been set to a network, then the network numbers field will be displayed in blue. The processing then stops in front of the network where the breakpoint is located. In this case, the network numbers field will become red. Using stepping (single step), you can jump from network to network.

The current value is displayed for each variable. Exception: If the input to a function block is an expression, only the first variable in the expression is monitored.

Doubleclicking on a variable opens the dialog box for writing a variable. Here it is possible to change the present value of the variable. In the case of Boolean variables, no dialog box appears; these variables are toggled.

The new value will turn red and will remain unchanged. If the **'Online' 'Write values'** command is given, then all variables are placed in the selected list and are once again displayed in black.

The flow control is started with the **'Online' 'Flow control'** command Using the flow control, you can view the present values that are being carried in the networks over the connecting lines. If the connecting lines do not carry Boolean values, then the value will be displayed in a specially inserted field. If the lines carry Boolean values, then they will be shaded blue in the event that they carry TRUE. By this means, you can accompany the flow of information while the PLC is running.

If you place the mouse pointer briefly above a variable, then the type, the address and the comment about the variable will be displayed in a Tooltip.

5.3.2 The Ladder Editor

This is how a POU written in the LD appears in the **CoDeSys** editor:

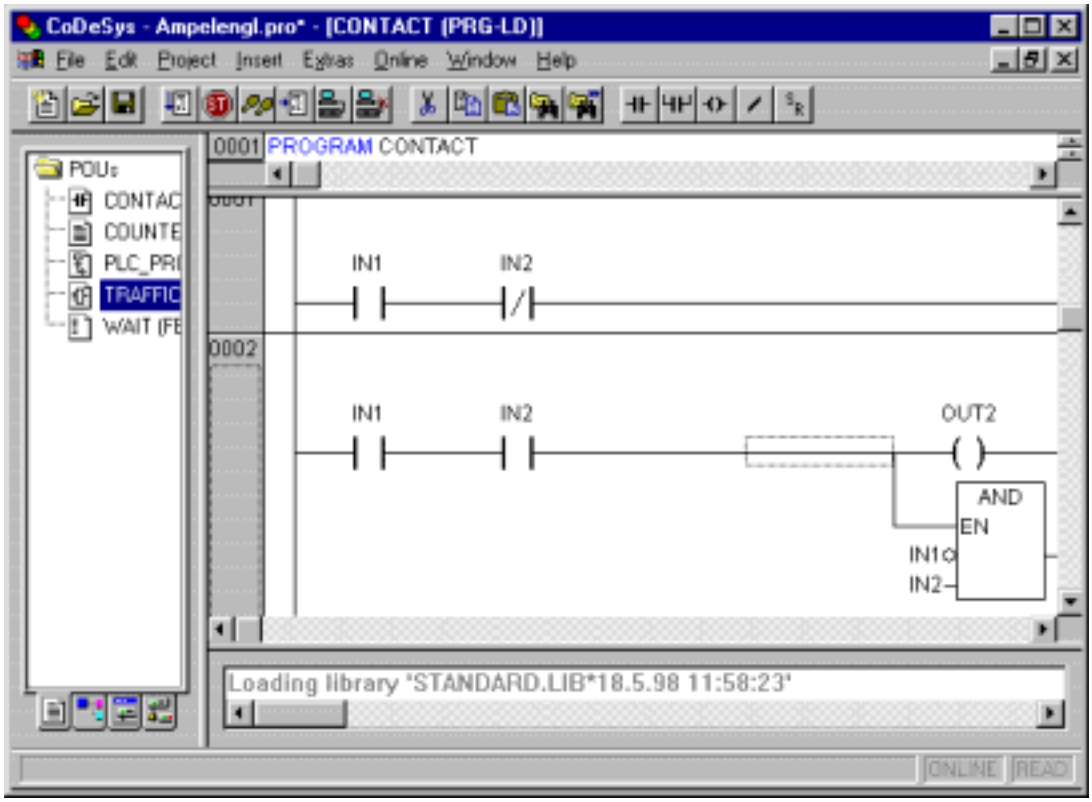


Image 5.12: POU in the Ladder Diagram

All editors for POUs consist of a declaration part and a body. These are separated by a screen divider.

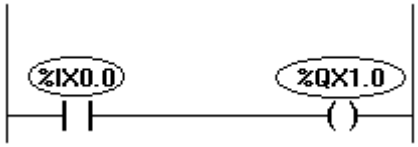
The LD editor is a graphic editor. The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

For information about the elements, see Chapter 2.2.6 Ladder Diagram (LD).

Cursor Positions in the LD Editors

The following locations can be cursor positions, in which the function block and program accessing can be handled as contacts. POUs with EN inputs and other POUs connected to them are treated the same way as in the Function Block Diagram. Information about editing this network part can be found in the Chapteron the FBD Editor.

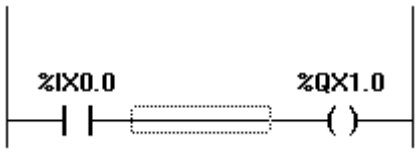
1. Every text field (possible cursor positions framed in black)



2. Every Contact or Function Block

3. Every Coil

4. The Connecting Line between the Contacts and the Coils.



The Ladder Diagram uses the following menu commands in a special way:

'Insert' 'Contact'

Symbol: **Shortcut: <Ctrl>+<O>**

Use this command in the LD editor in order to insert a contact in front of the marked location in the network.

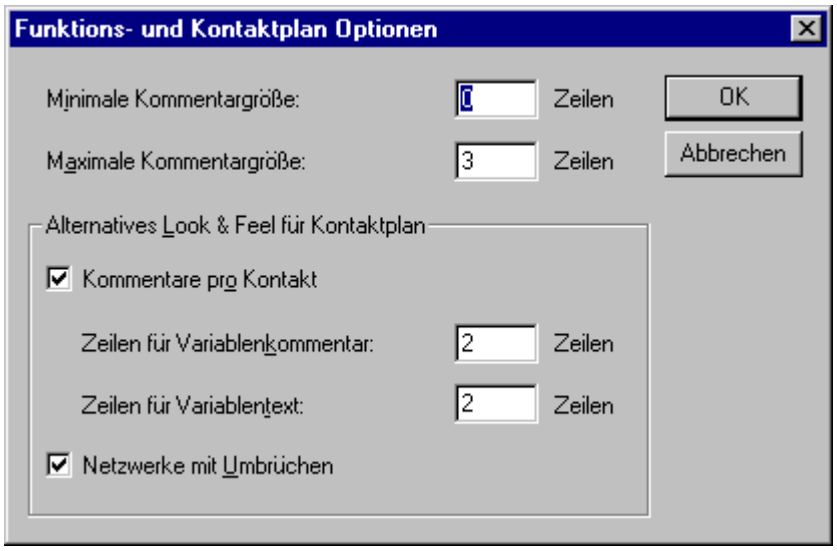
If the marked position is a coil (Cursor Position 3) or the connecting line between the contacts and the coils (Cursor Position 4), then the new contact will be connected serially to the previous contact connection.

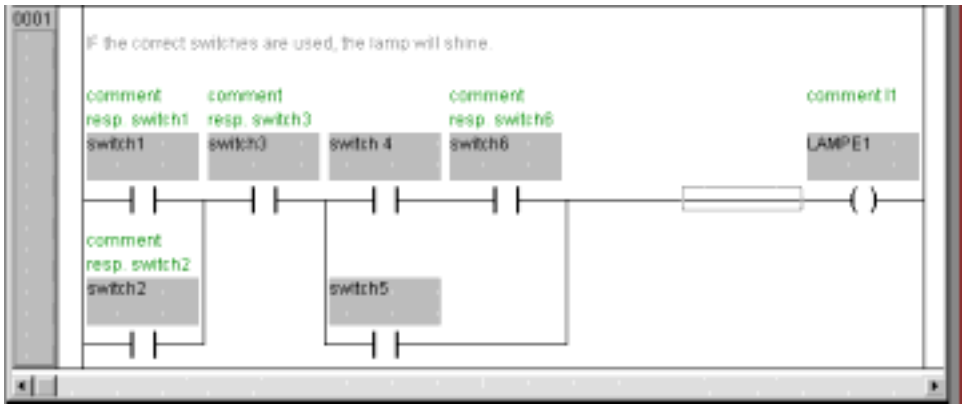
The contact is preset with the text "???". You can click on this text and change it to the desired variable or the desired constant. For this you can also use the Input Assistant.

You can activate the options **Comments per Contact** and **Lines for variable comment** in the dialog 'Function Block and Ladder Diagram Options' ('Extras' 'Options') to reserve a certain number of lines for the variable name. This might be useful, if long variable names are used, to keep the network short.

Also regard the option **Networks with linebreaks**, which you also can activate in the Ladder Diagram Options.

Example for the options dialog and the resulting display in a FBD or Ladder network:





'Insert' 'Parallel Contact'

Symbol: **Shortcut: <Ctrl>+<R>**

Use this command in the LD editor to insert a contact parallel to the marked position in the network.

If the marked position is a coil (Cursor Position 3) or the connection between the contacts and the coils (Cursor Position 4), then the new contact will be connected in parallel to the entire previous contact connection.

The contact is preset with the text "???". You can click on this text and change it to the desired variable or the desired constant. For this you can also use the Input Assistant.

It is possible to display the variable name with linebreaks. Also a separate comment can be inserted for the contact. For a description see 'Insert' 'Contact'.

'Insert' 'Function Block '

Shortcut: <Ctrl>+<B>

Use this command in order to insert an operator, a function block, a function or a program as a POU. For this, the connection between the contacts and the coils (cursor position 4), or a coil (cursor position 3), must be marked. The new POU at first has the designation AND. If you wish, you can change this designation to another one. For this you can also use the Input Assistant. Both standard and self-defined POUs are available.

The first input to the POU is placed on the input connection, the first output on the output connection; thus these variables must definitely be of type BOOL. All other in- and outputs of the POU are filled with the text „???". These prior entries can be changed into other constants, variables or addresses. For this you can also use the Input Assistant.

'Insert''Coil'

Symbol: **Shortcut: <Ctrl>+<L>**

You can use this command in the LD editor to insert a coil in parallel to the previous coils.

If the marked position is a connection between the contacts and the coils (Cursor Position 4), then the new coil will be inserted as the last. If the marked position is a coil (Cursor Position 3), then the new coil will be inserted directly above it.

The coil is given the text "???" as a default setting. You can click on this text and change it to the desired variable. For this you can also use the Input Assistant.

It is possible to display the variable name with linebreaks. Also a separate comment can be inserted for the coil. For a description see 'Insert' 'Contact'.

POUs with EN Inputs

If you want to use your LD network as a PLC for calling up other POUs , then you must merge a POU with an EN input. Such a POU is connected in parallel to the coils. Beyond such a POU you can develop the network further, as in the Function Block Diagram. You can find the commands for insertion at an EN POU under the menu item **'Insert' 'Insert at Blocks'**

An operator, a function block, a program or a function with EN input performs the same way as the corresponding POU in the Function Block Diagram, except that its execution is controlled on the EN input. This input is annexed at the connecting line between coils and contacts. If this connection carries the information "On", then the POU will be evaluated.

If a POU has been created once already with EN input, then this POU can be used to create a network. This means that data from usual operators, functions, and function blocks can flow in an EN POU and an EN POU can carry data to such usual POUs.

If, therefore, you want to program a network in the LD editor, as in FBD, you only need first to insert an EN operator in a new network. Subsequently, from this POU, you can continue to construct from your network, as in the FBD editor. A network thus formed will perform like the corresponding network in FBD.

‘Insert’ ’Box with EN ’

Use this command to insert a function block, an operator, a function or a program with EN input into a LD network.

The marked position must be the connection between the contacts and the coils (Cursor Position 4) or a coil (Cursor Position 3). The new POU is inserted in parallel to the coils and underneath them; it contains initially the designation "AND". If you wish, you can change this designation to another one. For this you can also use the Input Assistant.

’Insert’ ’Function Block with EN’

With this command you can insert a function block with EN input into a LD network.

The marked position must be the connection between the contacts and the coils (Cursor Position 4) or a coil (Cursor Position 3). The new function block is inserted in parallel to the coils, below them. From the Input Assistant dialog box that appears, you can select whether to insert a user-defined, or a standard (default) function block.

’Insert’ ’Function with EN’

With this command you can insert a function with EN input into an LD network.

The marked position must be the connection between the contacts and the coils (Cursor Position 4) or a coil (Cursor Position 3). The new function is inserted in parallel to the coils, below them. From the Input Assistant dialog box that appears, you can select whether to insert a user-defined, or a standard function block.

‘Insert’ ’Insert at blocks

With this command you can insert additional elements into a POU that has already been inserted (also a POU with EN input). The commands below this menu item can be executed at the same cursor positions as the corresponding commands in the Function Block Diagram (See Chapter 3.3.1).

With **Input** you can add a new input to the POU.

With **Output** you can add a new output to the POU.

With **POU**, you insert a new POU. The procedure is similar to that described under 'Insert' 'POU'.

With **Assign** you can insert an assignment to a variable. At first, this is shown by three question marks „???“, which you edit and replace with the desired variable. Input assistance is available for this purpose.

'Insert' 'Jump'

With this command you can insert a parallel jump in the selected LD editor, in parallel, at the end of the previous coils. If the incoming line delivers the value "On", then the jump will be executed to the indicated label.

The marked position must be the connection between the contacts and the coils(Cursor Position 4) or a coil (Cursor Position 3).

The jump is present with the text "???". You can click on this text and make a change in the desired label.

'Insert' 'Return'

In the LD editor, you can use this command to insert a Return instruction in parallel at the end of the previous coils. If the incoming line delivers the value "On," then the processing of the POU in this network is broken off.

The marked position must be the connection between the contacts and the coils(Cursor Position 4) or a coil (Cursor Position 3).

'Extras' 'Paste after'

Use this command in the LD editor to insert the contents of the clipboard as serial contact below the marked position. This command is only possible if the contents of the clipboard and the marked position are networks comprised of contacts.

'Extras' 'Paste below'

Shortcut: <Ctrl>+<U>

Use this command in the LD editor to insert the contents of the clipboard as parallel contact below the marked position. This command is only possible if the contents of the clipboard and the marked position are networks comprised of contacts.

'Extras' 'Paste above'

Use this command in the LD editor to insert the contents of the clipboard as parallel contact above the marked position. This command is only possible if the contents of the clipboard and the marked position are networks comprised of contacts.

'Extras' 'Negate'

Symbol: **Shortcut: <Strg>+<N>**

Use this command to negate a contact, a coil, a jump or return instruction, or an input or output of EN POUs at the present cursor position (Cursor Position 2 and 3).

Between the parentheses of the coil or between the straight lines of the contact, a slash will appear (**(/)** or |/|). If there are jumps, returns, or inputs or outputs of EN POUs, a small circle will appear at the connection, just as in the FBD editor.

The coil now writes the negated value of the input connection in the respective Boolean variable. Right at this moment, a negated contact switches the status of the input to the output, if the respective Boolean variable carries the value FALSE.

If a jump or a return is marked, then the input of this jump or return will be negated.

A negation can be canceled through renewed negation.

‘Extras’ ‘Set/Reset‘

If you execute this command on a coil, then you will receive a Set Coil. Such a coil never overwrites the value TRUE in the respective Boolean variable. This means that once you have set the value of this variable to TRUE, it will always remain at TRUE. A Set Coil is designated with an "S" in the coil symbol.

If you execute this command once again, then you will be given a Reset Coil. Such a coil never overwrites the value FALSE in the respective Boolean variable. This means that once you have set the value of this variable to FALSE, it will always remain at FALSE. A Reset Coil is designated with an "R" in the coil symbol.

If you execute this command repeatedly, the coil will alternate between set, reset and normal coil.

The Ladder Diagram in the Online Mode

In Online mode, the contacts and coils in the Ladder Diagram that are in the "On" state are colored blue. Likewise, all lines over which the "On" is carried are also colored blue. At the inputs and outputs of function blocks, the values of the corresponding variables are indicated.

Breakpoints can only be set on networks; by using stepping, you can jump from network to network.

If you place the mouse pointer briefly above a variable, then the type, the address and the comment about the variable will be displayed in a Tooltip.

5.3.3 The Sequential Function Chart Editor

The Sequential Function Chart editor is a graphic editor. The most important commands are found in the context menu (right mouse button or <Ctrl><F10>). Tooltips show in Offline as well as in Online mode and in the zoomed state the full names or expressions of steps, transitions, jumps, jump labels, qualifiers or associated actions.

For information about the Sequential Function Chart see also Chapter 2.2.3 .

See Image 5.13 for how a POU written in the SFC appears in the **CoDeSys** editor:

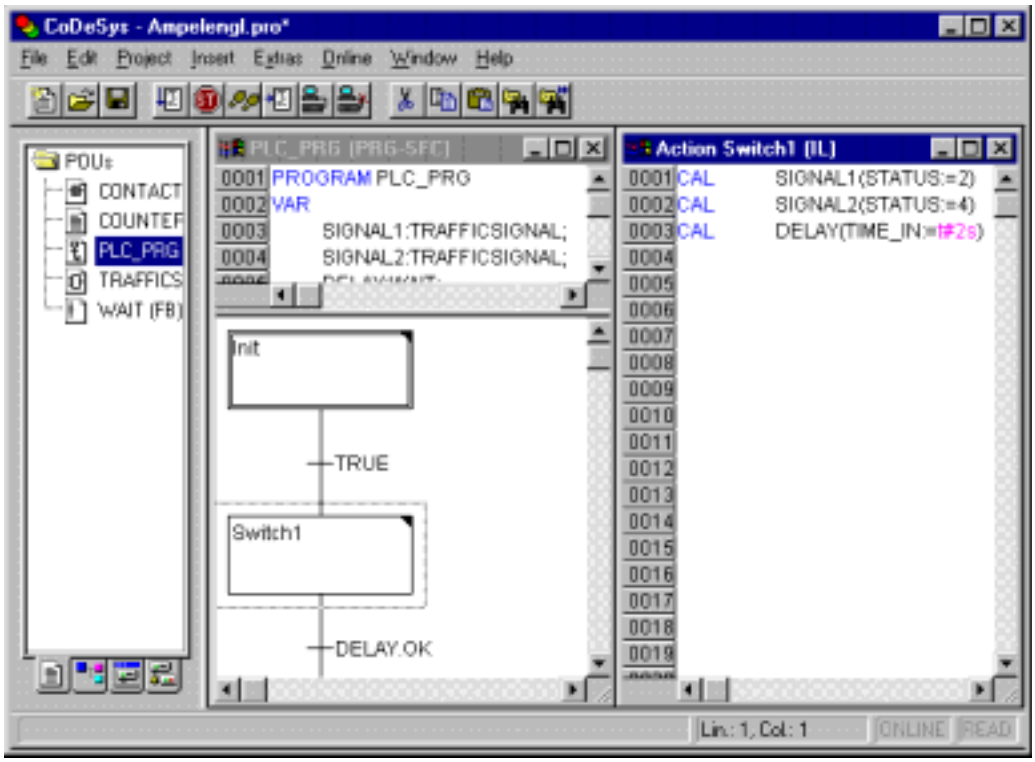


Image 5.13: Sequential Function Chart Editor with an opened Action

The editor for the Sequential Function Chart must agree with the particulars of the SFC. In reference to these, the menu items described in the following chapters will be of service:

Marking Blocks in the SFC

A marked block is a bunch of SFC elements that are enclosed in a dotted rectangle.

You can select an element (a step, a transition, or a jump) by pointing the mouse on this element and pressing the left mouse button, or you can use the arrow keys. In order to mark a group of several elements, press <Shift> for a block already marked, and select the element in the lower left or right corner of the group. The resulting selection is the smallest cohesive group of elements that includes both of these elements.

Please regard, that a step can only be deleted together with the preceeding or the succeeding transition !

Regard also that all commands can only be executed, if they do not contradict the conventions of the language.

'Insert' 'Step Transition (before)'

Symbol: **Shortcut: <Ctrl>+<T>**

This command inserts a step in the SFC editor followed by a transition in front of the marked block.

'Insert' 'Step Transition (after)'

Symbol: **Shortcut: <Ctrl>+<E>**

This command inserts a step in the SFC editor followed by a transition after the first transition in the marked block.

Delete Step and Transition

A step can only be deleted together with the preceeding or the succeeding transition. For this purpose put a selection frame around step and transition and choose command 'Edit' 'Delete' or press the <Del> key.

'Insert' 'Alternative Branch (right)'

Symbol: **Shortcut: <Ctrl>+<A>**

This command inserts an alternative branch in the SFC editor as a right branch of the marked block. For this the marked block must both begin and end with a transition. The new branch is then made up of one transition.

'Insert' 'Alternative Branch (left)'

Symbol:

This command inserts an alternative branch in the SFC editor as the left branch of the marked block. For this the marked block must both begin and end with a transition. The new branch is then made up of one transition.

'Insert' 'Parallel Branch (right)"

Symbol: **Shortcut: <Ctrl>+<L>**

This command inserts a parallel branch in the SFC editor as the right branch of the marked block. For this the marked block must both begin and end with a step. The new branch is then made up of one step. To allow jumps to the parallel branches that have been created, these must be provided with a jump label.

'Insert' 'Parallel Branch (left)'

Symbol:

'This command inserts a parallel branch in the SFC editor as the left branch of the marked block. For this the marked block must both begin and end with a step. The new branch is then made up of one step. To allow jumps to the parallel branches that have been created, these must be provided with a jump label (see 'Extras' 'Add label to parallel branch').

'Insert' 'Jump'

'This command inserts a jump in the SFC editor at the end of the branch, to which the marked block belongs. For this the branch must be an alternative branch.

The inserted text string 'Step' in the inserted jump can then be selected and replaced by the step name or the jump label of a parallel branch to be jumped to.

’Insert’ ’Transition-Jump’

Symbol:

This command inserts a transition in the SFC editor, followed by a jump at the end of the selected branch. For this the branch must be a parallel branch.

The inserted text string 'Step' in the inserted jump can then be selected and replaced by the step name or the jump label of a parallel branch to be jumped to.

’Insert’ ’Add Entry-Action’

With this command you can add an entry-action to a step. An entry-action is only executed once, right after the step has become active. The entry-action can be implemented in a language of your choice.

A step with an entry-action is designated by an "E" in the bottom left corner.

’Insert’ ’Add Exit-Action’

With this command you can add an exit-action to a step. An exit-action is only executed once, before the step is deactivated. The exit-action can be implemented in a language of your choice.

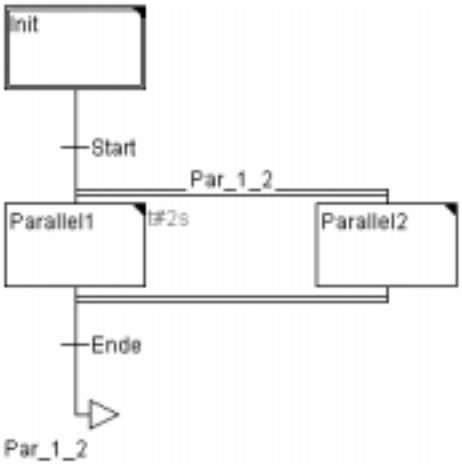
A step with an exit-action is designated by an "X" in the lower right corner.

'Extras' 'Paste Parallel Branch (right)'

This command pastes the contents of the clipboard as a right parallel branch of the marked block. For this the marked block must both begin and end with a step. The contents of the clipboard must, likewise, be an SFC block that both begins and ends with a step.

'Extras' 'Add label to parallel branch'

In order to provide a newly inserted parallel branch with a jump label, the transition occurring before the parallel branching must be marked and the command 'Add label to parallel branch' must be executed. At that point, the parallel branch will be given a standard name consisting of „Parallel“ and an appended serial number, which can be edited according to the rules for identifier names. In the following example, "Parallel" was replaced by "Par_1_2" and the jump to the transition "End" was steered to this jump label.



Delete a label

A jump label can be deleted by deleting the label name.

'Extras' 'Paste after'

This command pastes the SFC block on the clipboard after the first step or the first transition of the marked block. (Normal copying pastes it in front of the marked block.) This will now be executed, if the resulting SFC structure is correct, according to the language norms.

'Extras' 'Zoom Action/Transition'

Shortcut: <Alt>+<Enter>

The action of the first step of the marked block or the transition body of the first transition of the market block is loaded into the editor in the respective language, in which it has been written. If the action or the transition body is empty, then the language must be selected, in which it has been written.

'Extras' 'Clear Action/Transition'

With this command you can delete the actions of the first step of the marked block or of the transitions body of the first transition.

If, during a step, you implement either only the action, the entry-action, or the exit-action, then the same will be deleted by the command. Otherwise a dialog box appears, and you can select which action or actions are to be deleted.

If the cursor is located in the action of an IEC step, then only this association will be deleted. If an IEC step with an associated action is selected, then this association will be deleted. During an IEC step with several actions, a selection dialog box will appear.

'Extras' 'Step Attributes'

With this command you can open a dialog box in which you can edit the attributes for the marked step.

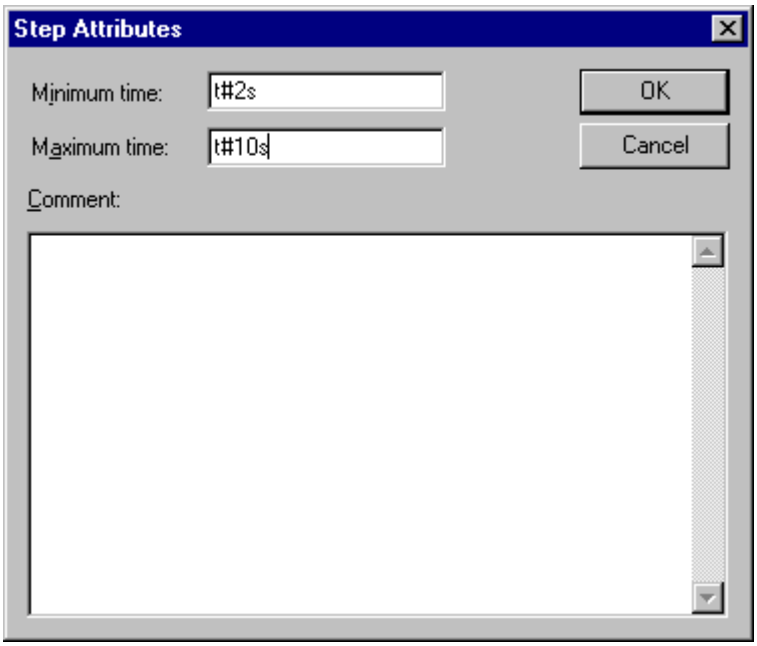
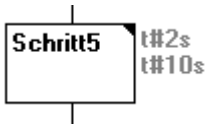


Image 5.14: Dialog Box for Editing Step Attributes

You can take advantage of three different entries in the step attribute dialog box. Under **Minimum Time**, you enter the minimum length of time that the processing of this step should take. Under the **Maximum Time**, you enter the maximum length of time that the processing of this step should take. Note that the entries are of the **TIME** type, so you use a TIME constant (i.e. T#3s) or a variable of the TIME type.

Under **Comment** you can insert a comment to the step. In the 'Sequential function chart options' dialog which you open under 'Extras' 'Options', you can then enter whether comments or the time setting is displayed for the steps in the SFC editor. On the right, next to the step, either the comment or the time setting will appear.

If the maximum time is exceeded, SFC flags are set which the user can query.



The example shows a step whose execution should last at least two, and at the most, ten seconds. In Online mode, there is, in addition to these two times, a display of how long the step has already been active.

'Extras' 'Time Overview'

With this command you can open a window in which you can edit the time settings of your SFC steps:

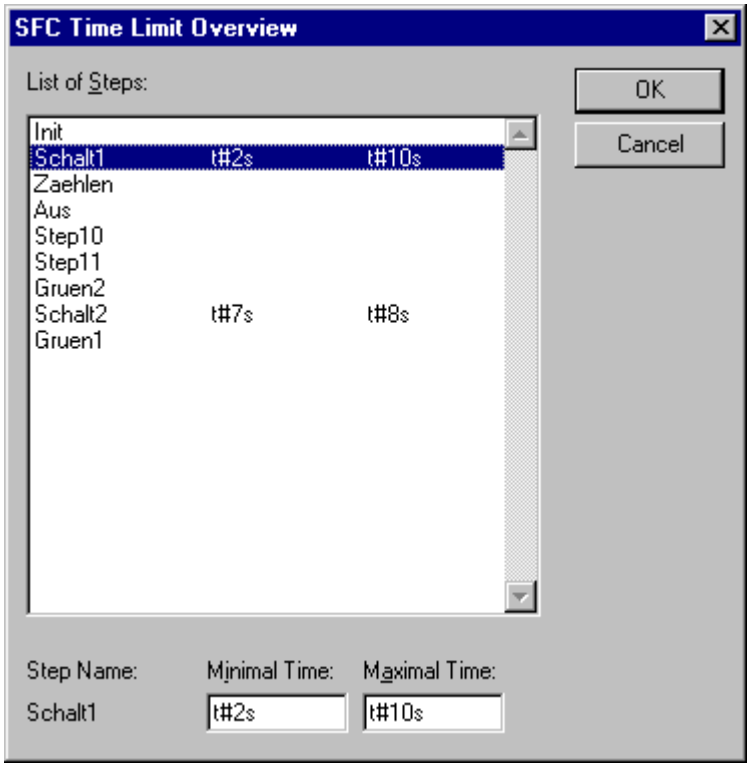


Image 5.15: Time Boundaries Overview for a SFC POU

In the time boundaries overview, all steps of your SFC POU are displayed. If you have entered a time boundary for a step, then the time boundary is displayed to the right of the step (first, the lower limit, then the upper limit). You can also edit the time boundaries. To do so, click on the desired step in the overview. The name **of the step** is then shown below in the window. Go to the **Minimum Time** or **Maximum Time** field, and enter the desired time boundary there. If you close the window with **OK**, then all of the changes will be stored.

In the example, steps 2 and 6 have a time boundary. Shift1 lasts at least two, and at most, ten seconds. Shift2 lasts at least seven, and at most, eight seconds.

'Extras' 'Options'

With this command you open a dialog box in which you can set different options for your SFC POU.

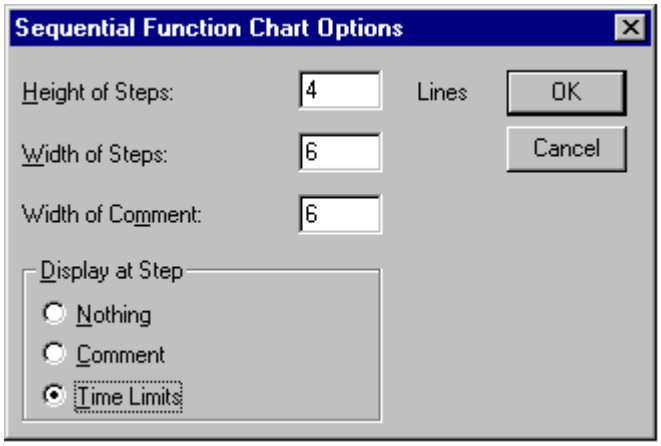


Image 5.16: Dialog Box for Sequential Function Chart Options

In the SFC Options dialog box you can undertake five entries. Under **Step Height**, you can enter how many lines high an SFC step can be in your SFC editor. 4 is the standard setting here. Under **Step Width**, you can enter how many columns wide a step should be. 6 is the standard setting here. You can also preset the **Display at Step**. With this, you have three possibilities: You can either have **Nothing** displayed, or the **Comment**, or the **Time Limits**. The last two are shown the way you entered them in **'Extras' 'Step Attributes'**.

'Extras' 'Associate Action'

With this command actions and Boolean variables can be associated with IEC steps.

To the right of, and next to the IEC step, an additional divided box is attached, for the association of an action. It is preset in the left field with the qualifier "N" and the name "Action." Both presets can be changed. For this you can use the Input Assistant.

Maximum nine actions can be assigned to an IEC step.

New actions for IEC steps are created in the Object Organizer for an SFC POU with the **'Project' 'Add Action'** command.

'Extras' 'Use IEC-Steps'

Symbol:

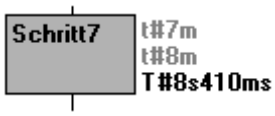
If this command is activated (denoted by a check in front of the menu item and a printed symbol in the Tool bar), then IEC steps will be inserted instead of the simplified steps upon insertion of step transitions and parallel branches.

If this option is switched on, the Init step is set as an IEC step when you create a new SFC POU.

This settings are saved in the file "CoDeSys.ini" and are restored when CoDeSys gets started again.

Sequential Function Chart in Online Mode

With the Sequential Function Chart editor in Online mode, the currently active steps will be displayed in blue. If you have set it under **'Extras' 'Options'**, then the time management is depicted next to the steps. Under the lower and upper bounds that you have set, a third time indicator will appear from which you can read how long the step has already been active.



In the picture above the step depicted has already been active 8 seconds and 410 milliseconds. The step must, however, be active for at least 7 minutes before the step will be left.

With **'Online' 'Toggle Breakpoint'** a breakpoint can be set on a step, or in an action at the locations allowed by the language in use. Processing then stops prior to execution of this step or before the location of the action in the program. Steps or program locations where a breakpoint is set are marked in light blue.

If several steps are active in a parallel branch, then the active step whose action will be processed next is displayed in red.

If IEC steps have been used, then all active actions in Online mode will be displayed in blue.

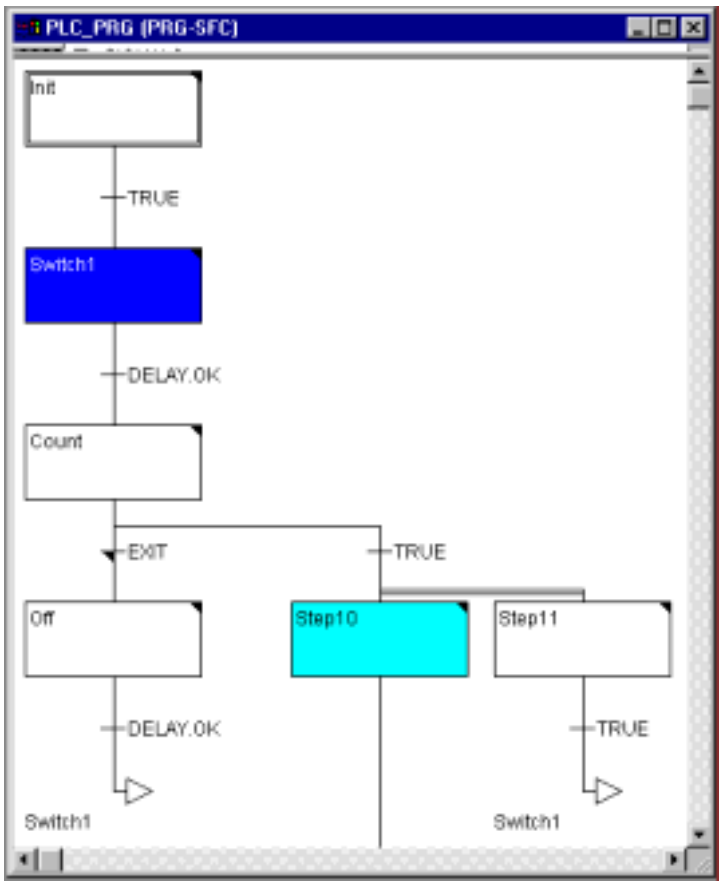


Image 5.17: Sequential Function Chart in the Online Mode with an Active Step (Shift1) and a Breakpoint (Step10)

With the command **'Online' 'Step over'** it is stepped always to the next step which action is executed. If the current location is:

- a step in the linear processing of a POU or a step in the rightmost parallel branch of a POU, execution returns from the SFC POU to the caller. If the POU is the main program, the next cycle begins.
- a step in a parallel branch other than the rightmost, execution jumps to the active step in the next parallel branch.
- the last breakpoint location within a 3S action, execution jumps to the caller of the SFC.
- the last breakpoint location within an IEC action, execution jumps to the caller of the SFC.
- the last breakpoint position within an input action or output action, execution jumps to the next active step.

With **'Online' 'Step in'** even actions can be stepped into. If an input, output or IEC action is to be jumped into, a breakpoint must be set there. Within the actions, all the debugging functionality of the corresponding editor is available to the user.

If you rest the mouse cursor for a short time on a variable in the declaration editor, the type, the address and the comment of the variable will be displayed in a **tooltip**.

Please regard: If you rename a step and perform an Online Change while this step is active, the program will be stopped in undefined status !

Processing order of elements in a sequence:

1. First, all Action Control Block flags in the IEC actions that are used in this sequence are reset (not, however, the flags of IEC actions that are called within actions).
2. All steps are tested in the order which they assume in the sequence (top to bottom and left to right) to determine whether the requirement for execution of the output action is provided, and this is executed if that is the case.
3. All steps are tested in the order which they assume in the sequence to determine whether the requirement for the input action is provided, and this is executed if that is the case.
4. For all steps , the following is done in the order which they assume in the sequence:
   - If applicable, the elapsed time is copied into the corresponding step variable.
   - If applicable, any timeout is tested and the SFC error flags are serviced as required.
   - For non-IEC steps, the corresponding action is now executed.
5. IEC actions that are used in the sequence are executed in alphabetical order. This is done in two passes through the list of actions. In the first pass, all the IEC actions that are deactivated in the current cycle are executed. In the second pass, all the IEC actions that are active in the current cycle are executed.
6. Transitions are evaluated: If the step in the current cycle was active and the following transition returns TRUE (and if applicable the minimum active time has already elapsed), then the following step is activated.

The following must be noted concerning implementation of actions:

It can come about that an action is carried out several times in one cycle because it is associated with multiple sequences. (For example, an SFC could have two IEC actions A and B, which are both implemented in SFC, and which both call IEC action C; then in IEC actions A and B can both be active in the same cycle and furthermore in both actions IEC action C can be active; then C would be called twice).

If the same IEC action is used simultaneously in different levels of an SFC, this could lead to undesired effects due to the processing sequence described above. For this reason, an error message is issued in this case. It can possibly arise during processing of projects created with older versions of **CoDeSys**.

Note: In monitoring expressions (e.g. A AND B) in transitions, only the „Total value“ of the transition is displayed.

5.3.4 The Continuous Function Chart Editor (CFC)

It looks like a block which has been produced using the continuous function chart editor (CFC):

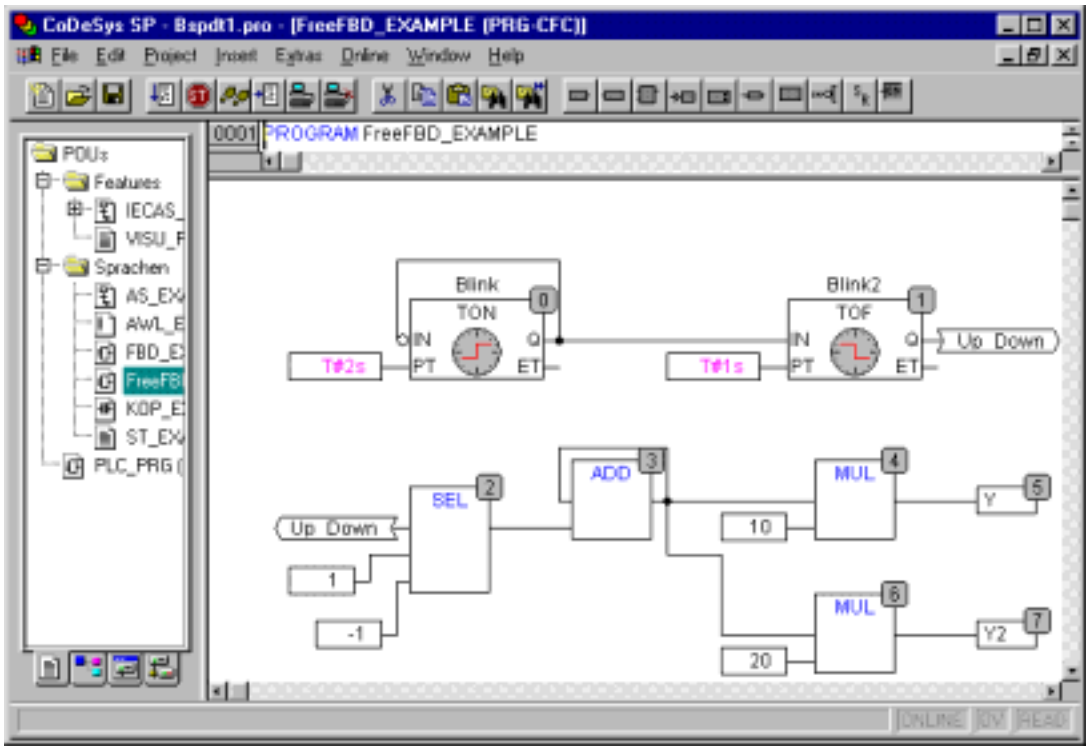


Image 5.18: Editor for continuous function charts

No snap grid is used for the continuous function chart editor so the elements can be placed anywhere. Elements of the sequential processing list include boxes, input, output, jump, label, return and comments. The inputs and outputs of these elements can be connected by dragging a connection with the mouse. The connecting line will be drawn automatically. The shortest possible connection line is drawn taking into account existing connections. The connecting lines are automatically adjusted when the elements are moved. If the case arises where a connecting line cannot be drawn simply because of lack of space, a red line will be shown between the input and the associated output instead. This line will be converted into a connecting line just as soon as space is available.

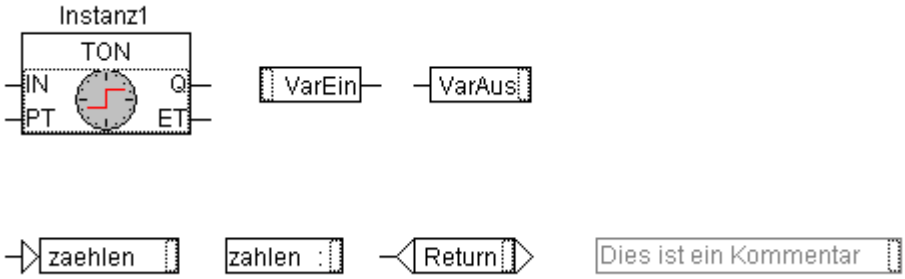
One advantage of the continuous function chart as opposed to the usual function block diagram editor FBD is the fact that feedback paths can be inserted directly.

The most important commands can be found in the context menu

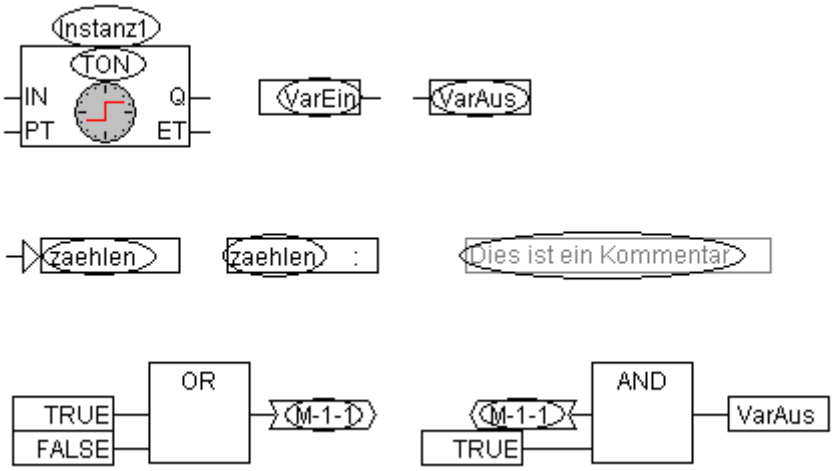
Cursor positions Each text is a possible cursor position. The selected text is shaded in blue and can be modified.

In all other cases the current cursor position is shown by a rectangle made up of points. The following is a list of all possible cursor positions with examples:

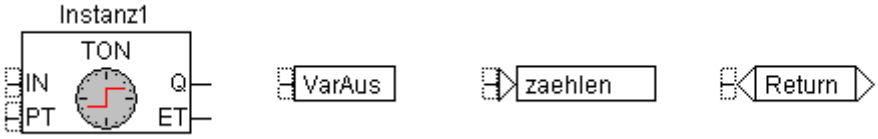
1. Trunks of the elements box, input, output, jump, label, return and comments.



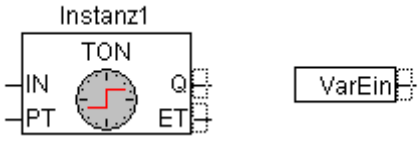
2. Text fields for the elements box, input, output, jump, label, return and comments as well as text fields for connection marker



3. Inputs for the elements box, input, output, jump and return



4. Outputs for the elements box and input:



'Insert' 'Box'

Symbol: **Shortcut: <Ctrl>+<B>**

This command can be used to paste in operators, functions, function blocks and programs First of all, it is always inserted an "AND" operator. This can be converted by Selection and Overwrite of the text into every other operator, into every function, into every function block and every program. The input assistance serves to select the desired block from the list of supported blocks. If the new block has another minimum number of inputs, these will be attached. If the new block has a smaller highest number of inputs, the last inputs will be deleted.

‚Insert‘ ‚Input

Symbol: **Shortcut: <Ctrl> + <E>**

This command is used to insert an input. The text offered ”???” can be selected and replaced by a variable or constant. The input assistance can also be used here.

‘Insert’ ‘Output’

Symbol: **Shortcut: <Ctrl>+<A>**

This command is used to insert an output. The text offered ”???” can be selected and replaced by a variable. The input assistance can also be used here. The value which is associated with the input of the output is allocated to this variable.

‘Insert’ ‘Jump‘

Symbol: **Shortcut: <Ctrl>+<J>**

This command is used to insert a jump. The text offered ”???” can be selected and replaced by the jump label to which the program should jump. The jump label is inserted using the command **‘Insert ‘Label'**

‘Insert’ ‘Label‘

Symbol: **Shortcut: <Ctrl>+<L>**

This command is used to insert a label. The text offered ”???” can be selected and replaced by the jump label. In Online mode a RETURN label for marking the end of POU is automatically inserted. The jump is inserted using the command **‘Insert ‘Jump'**

‘Insert’ ‘Return’

Symbol: **Shortcut: <Ctrl> + <R>**

This command is used to insert a RETURN command. Note that in Online mode a jump label with the name RETURN is automatically inserted in the first column and after the last element in the editor; in stepping, it is automatically jumped to before execution leaves the POU.

‘Insert’ ‘Comment’

Symbol: **Shortcut: <Ctrl> + <K>**

This command is used to insert a comment.

You obtain a new line within the comment with <Ctrl> + <Enter>.

‘Insert’ ‘Input of box’

Shortcut: <Ctrl> + <U>

This command is used to insert an input at a box. The number of inputs is variable for many operators (e.g. ADD can have two or more inputs).

To increase the number of inputs for such an operator by one, the box itself must be selected (Cursor position 1)

Insert' 'In-Pin', 'Insert' 'Out-Pin'

Symbol:

These commands are available as soon as a macro is opened for editing. They are used for inserting in- or out-pins as in- and outputs of the macro. They differ from the normal in- and outputs of POUs by the way they are displayed and in that they have no position index.

'Extras' 'Negate'

Symbol: **Shortcut: <Ctrl> + <N>**

This command is used to negate inputs, outputs, jumps or RETURN commands. The symbol for the negation is a small cross on the connection.

The input of the element block, output, jump or return is negated when it is selected (Cursor position 3).

The output of the element block or input is negated when it is selected (Cursor position 4).

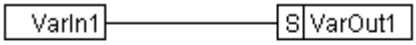
A negation can be deleted by negating again.

'Extras' 'Set/Reset'

Symbol: **Shortcut: <Ctrl> + <T>**

This command can only be used for selected inputs of the element output (Cursor position 3).

The symbol for Set is S and for Reset is R.



VarOut1 is set to TRUE, if VarIn1 delivers TRUE. VarOut1 retains this value, even when VarIn1 springs back to FALSE.

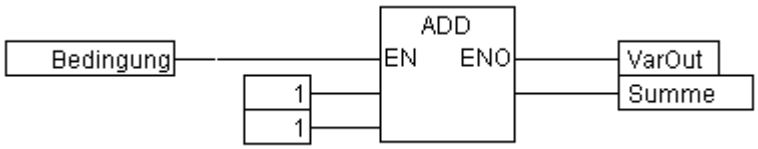
VarOut2 is set to FALSE, if VarIn2 delivers TRUE. VarOut2 retains this value, even when VarIn2 springs back to FALSE.

Multiple activation of this command causes the output to change between Set, Reset and the normal condition.

'Extras' 'EN/ENO'

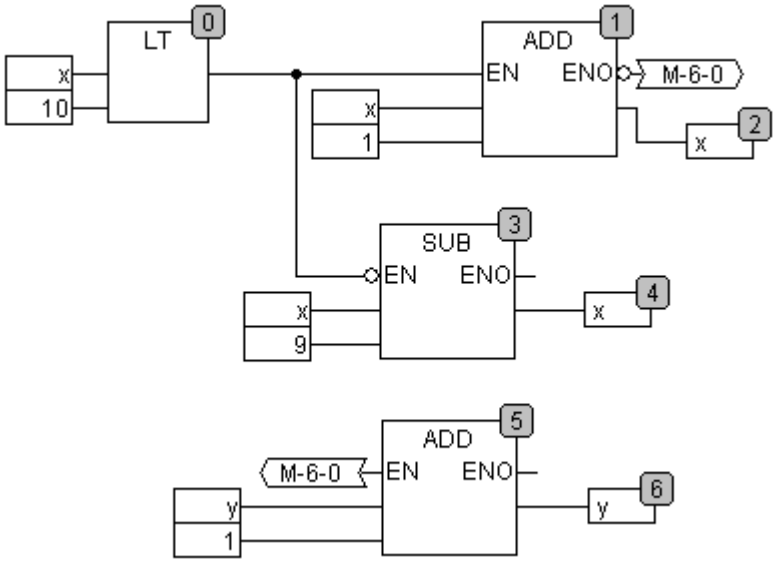
Symbol: **Shortcut: <Ctrl> + <0>**

This command is used to give a selected block (Cursor position 3) an additional Boolean enable input EN (Enable In) and a Boolean output ENO (Enable Out).



ADD is only executed in this example when the Boolean variable “Bedingung” (condition) is TRUE. VarOut is also set to TRUE after ADD has been executed. ADD will not be executed when the variable “Bedingung” (condition) is FALSE and VarOut retains its value FALSE.

The example below shows how the value ENO can be used for further blocks.



x should initialised to 1 and y initialised to 0. The numbers in the right corner of the block indicate the order in which the commands are executed.

x will be increased by one until it reaches the value 10. This causes the output of the block LT(0) to deliver the value FALSE and SUB(3) and ADD(5) will be executed. x is set back to the value 1 and y is increased by 1. LT(0) is executed again as long as x is smaller than 10. y thus count the number of times x passes though the value range 1 to 10.

‚Extras‘ ‚Properties...‘

Constant input parameters (VAR_INPUT CONSTANT) from functions and function blocks are not shown directly in the continuous function chart editor. These can be shown and their value can be changed when one selects the trunk of the block in question (Cursor position 1) and then selects the command ‚Extras‘ ‚Properties‘ or simply double clicks on the trunk. The „Edit parameters“ dialog opens:

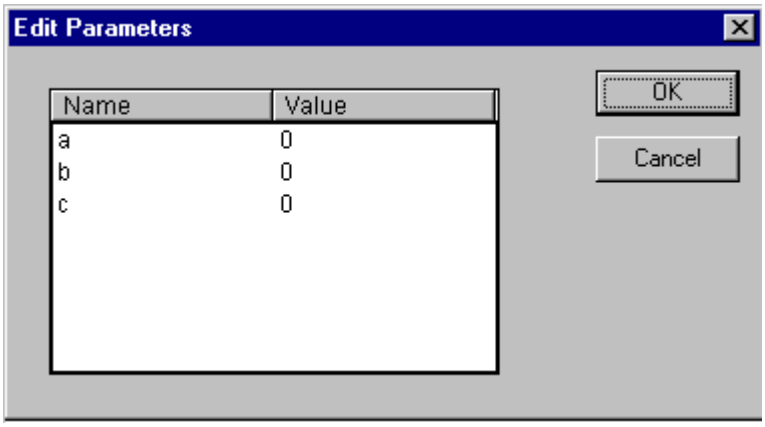


Image 5.19: Properties dialog

The values of the constant input parameter (VAR_INPUT CONSTANT) can be changed. Here it is necessary to mark the parameter value in the column Value. Another mouse click or pressing on the space bar allows this to be edited. Confirmation of the change to the value is made by pressing the <Enter> key or pressing <Escape> rejects the changes. The button **OK** stores all of the changes which were made.

Selecting elements

One clicks on the trunk of the element (Cursor position 1) to select it.

To mark more elements one presses the <Shift> key and clicks in the elements required, one after the other, or one drags the mouse with the left hand mousekey depressed over the elements to be marked.

The command **'Extras' 'Select all'** marks all elements at once.

Moving elements

One or more selected elements can be moved with the arrow keys as one is pressing on the <Shift> key. Another possibility is to move elements using a depressed left mousekey. These elements are placed by releasing the left mousekey in as far as they do not cover other elements or exceed the foreseen size of the editor. The marked element jumps back to its initial position in such cases and a warning tone sounds.

Copying elements

One or more selected elements can be copied with the command **'Edit' 'Copy'** and inserted with the command **'Edit' 'Paste'**.

Creating connections

An input of an element can be precisely connected to the output of another element. An output of an element can be connected to the inputs of a number of other elements.

There are a number of possibilities to connect the input of an element E2 with the output of an element E1.

Place the mouse on the output of element E1 (Cursor position 4), click with the left mousekey, hold the left mousekey down and drag the mouse cursor onto the input of element E2 (Cursor position 3) and let the left mousekey go. A connection is made from the output of element E1 to the mouse cursor during this dragging operation with the mouse.

Place the mouse on the input of element E2, click with the left mousekey, hold the left mousekey down and drag the mouse cursor onto the output of element E1 and let the left mousekey go.

Move one of the elements E1 or E2 (Cursor position 1) and place it in such a way by letting go of the left mousekey that the output of element E2 and the input of element E1 touch.

Where element E2 is a block with a free input, a connection can also be made by dragging the mouse from an output from E1 to the trunk of E2. A connection with the free input at the highest position on E2 will be created when the mousekey is released. In the case where block E2 does not have a free input but is an operator which can have an input added to it, a new input will be automatically generated.

The output and input of a block can be connected together (feedback path) by using this method. To establish a connection between two pins, click with the left mouse button on one pin, hold the button down and thus drag the connection to the desired pin, where you then release the button. If during the dragging of the connection extends outside working area of the editor, scrolling occurs automatically. For simple data types, type testing is carried out during the connection. If the types of the two pins are not compatible, the cursor changes to „Forbidden“. For complex data types, no testing takes place.

Deleting connections

There are a number of possibilities for removing the connection between the output of an element E1 and the input of an element E2:

Select the output of element E1 (Cursor position 4) and press the <Delete> key or execute the command **‘Edit’ ‘Delete'**. Several connections will be removed at the same if the output of E1 is connected to more than one of inputs.

Select the input of element E2 (Cursor position 4) and press the <Delete> key or execute the command **‘Edit’ ‘Delete‘**.

Select the input of E2 with the mouse, hold the left mousekey depressed and drag the connection from the input to E2 away. The connection is removed when the left mousekey is released in a free area of the screen.

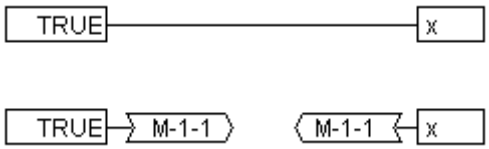
Changing connections

A connection between the output of an element E1 and the input of an element E2 can easily be changed into a connection between the output of element E1 and the input of element E3. The mouse is clicked on the input of E2 (Cursor position 3), the left mousekey is kept depressed, the mouse cursor is moved to the input of E3 and then released.

‘Extras’ ‘Connection marker‘

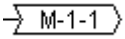
Connections can also be represented by a connector (connection marker) instead of a connecting line. Here the output and the associated input have a connector added to them which is given a unique name.

Where a connection already exists between the two elements which should now be represented by connectors, the output of the connecting line is marked (Cursor position 3) and the menu point ‚Extras‘ ‚Connection marker‘ is selected. The following diagram shows a connection before and after the selection of this menu point.

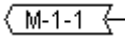


A unique name is given as standard by the program which begins with M, but which can be changed The connector name is stored as an output parameter, but can be edited both at the input and at the output.

It is important to know that the connector name is associated with a property of the output of a connection and is stored with it.

1. Edit the connector at the output: 

If the text in the connector is replaced, the new connector name is adopted by all associated connectors at the inputs. One cannot, however, select a name which already belongs to **another** connection marker since the uniqueness of the connector name would be violated.

2. Edit the connector at the input: 

If the text in a connector is replaced, it will also be replaced in the corresponding connection marker on the other POU. Connections in connector representations can be converted to normal connections in that one marks the output of the connections (Cursor position 4) and again selects the menu point **’Extras‘ ‘Connection marker‘**.

Insert inputs/outputs “on the fly”

If exactly one input or output pin of an element is selected, then the corresponding input- or output- element can be directly inserted and its editor field filled with a string by entering the string at the keyboard.

Order of execution

The elements block, output, jump, return and label each possess a number indicating the order in which they are executed. In this sequential order the individual elements are evaluated at run time.

When pasting in an element the number is automatically given according to the topological sequence (from left to right and from above to below). The new element receives the number of its topological successor if the sequence has already been changed and all higher numbers are increased by one.

The number of an element remains constant when it is moved.

The sequence influences the result and must be changed in certain cases.

If the sequence is displayed, the corresponding sequential execution number is shown in the upper right hand corner of the element.

‘Extras‘ ’Order‘ ’Display‘

This command switches the display of the order of execution on and off. The default setting is to show it (recognised by a tick (✓) in front of the menu point).

The relevant order of execution number appears in the upper right hand corner for the elements block, output, jump, return and label.

‘Extras‘ ’Order‘ ’Order topologically‘

Elements are ordered in a topological sequence when the execution takes place from left to right and from above to below, that is the number increases from left to right and from above to below for topologically arranged elements. The connections are not relevant, only the location of the elements is important.

All **selected** elements are topologically arranged when the command **‘Extras‘ ‘Order‘ ’Order topologically‘** is executed. All elements in the selection are taken out of the sequential processing list by this process. The elements are then entered into the remaining sequential processing list individually from bottom right through to upper left. Each marked element is entered into the sequential processing list before its topological successor, i.e. it is inserted before the element that in a topological sequencing would be executed after it, when all elements in the editor were sequenced according to a topological sequencing system. This will be clarified by an example.

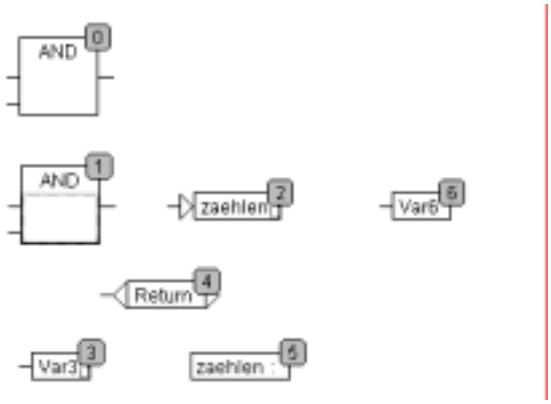


Image 5.20: Sequence before the topological ordering of the three elements

The elements with numbers 1, 2 and 3 are selected. If the command **’Order topologically‘** is selected the elements are first taken out of the sequential processing list. Var3, the jump and the AND-operator are then inserted again one after the other. Var3 is placed before the label and receives the number 2. The jump is then ordered and receives the number 4 at first but this then becomes 5 after the AND is inserted. The new order of execution which arises is:

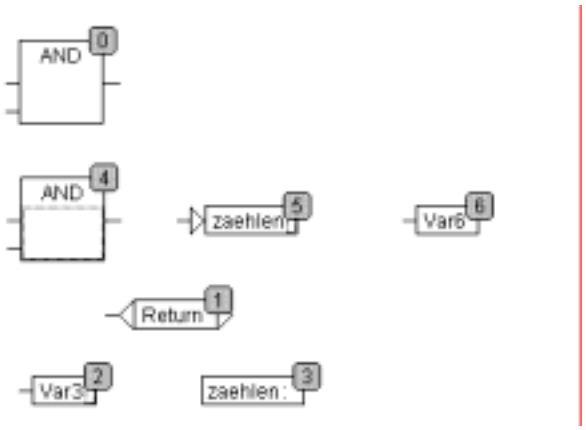


Image 5.21: Sequence after the topological ordering of the three elements

When a newly generated block is introduced it will be placed by default in front of its topological successor in the sequential processing list.

'Extras' 'Order' 'One forwards'

With this command all selected elements with the exception of the element which is at the beginning of the sequential processing list are moved one place forwards in the sequential processing list.

'Extras' 'Order' 'One backwards'

With this command all selected elements with the exception of the element which is at the end of the sequential processing list are moved one place backwards in the sequential processing list.

'Extras' 'Order' 'To the beginning'

With this command all selected elements will be moved to the front of the sequential processing list whereby the order within the group of selected elements is maintained. The order within the group of unselected elements also remains the same.

'Extras' 'Order' 'To the end'

With this command all selected elements will be moved to the end of the sequential processing list whereby the order within the group of selected elements is maintained. The order within the group of unselected elements also remains the same.

'Extras' 'Order' 'Order everything according to data flow''

This command **effects all** elements. The order of execution is determined by the data flow of the elements and not by their position. The diagram below shows elements which have been ordered topographically.

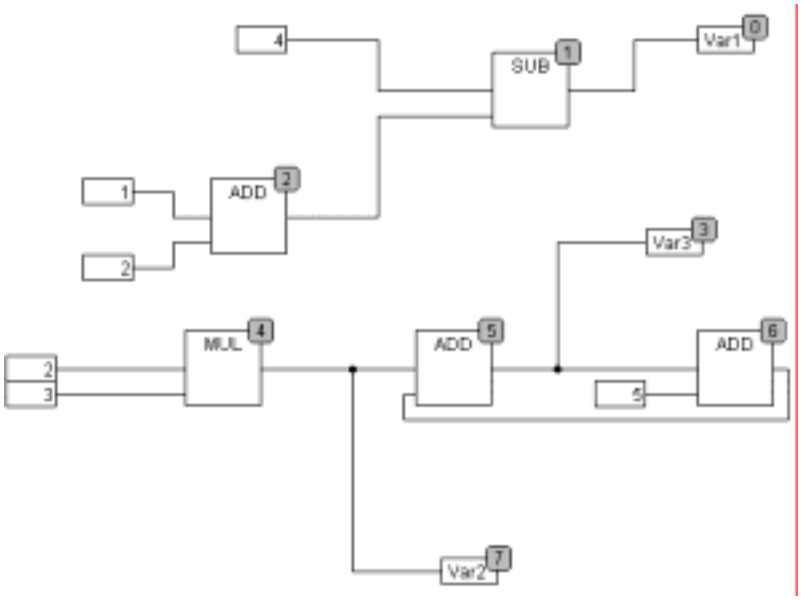


Image 5.22: Sequence before the ordering according to data flow

The following arrangement exists after selecting the command:

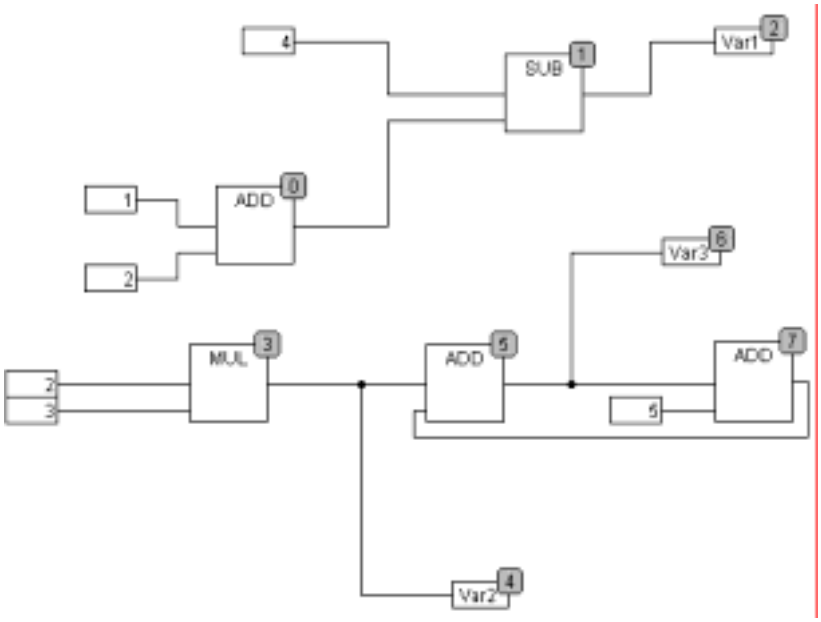


Image 5.23: Sequence after the ordering according to data flow

When this command is selected the first thing to happen is that the elements are ordered topographically. A new sequential processing list is then created. Based on the known values of the inputs, the computer calculates which of the as yet not numbered elements can be processed next. In the above ”network” the block AND, for example, could be processed immediately since the values at its inputs (1 and 2) are known. Block SUB can only then be processed since the result from ADD must be known first, etc.

Feedback paths are inserted last.

The advantage of the data flow sequencing is that an output box which is connected to the output of a block comes immediately after it in the data flow sequencing system which by topological ordering would not always be the case. The topological

ordering can deliver another result in some cases than ordering by data flow, a point which one can recognise from the above example.

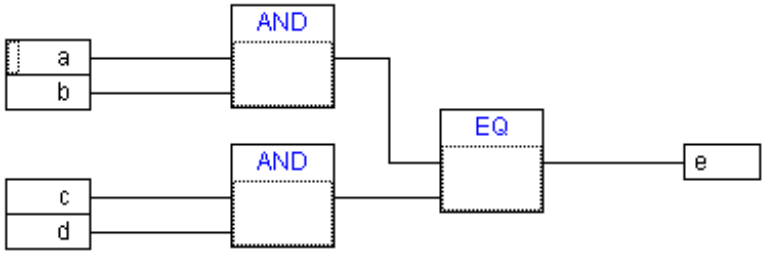
'Extras' 'Create macro'‘

With this command, several POUs that are selected at the same time can be assembled into a block, which can named as a macro. Macros only can be reproduced by Copy/Paste, whereby each copy becomes a separate macro whose name can be chosen independently. Macros are thus not references. All connections that are cut by the creation of a macro generate in- or out-pins on the macro. Connections to inputs generate an in-pin. The default name appears next to the pin in the form In<n>. For connections to outputs, Out<n> appears. Affected connections which had connection markers prior to the creation of the macro, retain the connection marker on the PIN of the macro.

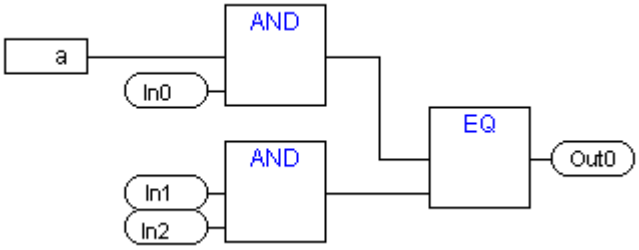
At first, a macro has the default name "MACRO". This can be changed in the Name field of the macro use. If the macro is edited, the name of the macro will be displayed in the title bar of the editor window appended to the POU name.

Example:

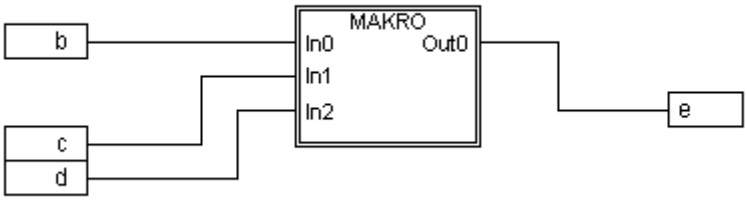
Selection



Macro:



In the editor:



'Extras' 'Edit Macro'

By this command, or by double clicking on the body of the macro, the macro is opened for editing in the editor window of the associated POU. The name of the macro is displayed appended to the POU name in the title bar.

The pin boxes generated for the in- and outputs of the macro during creation can be handled like normal POU in- and outputs. They can also be moved, deleted, added, etc. They differ only in how they are displayed and have no position index. Pin boxes have rounded corners. The text in the pin-box matches the name of the pin in the macro display.

The order of the pins in the macro box follows the order of execution of the elements of the macro. A lower order index before a higher one, higher pin before lower.

The processing order within the macro is closed, in other words the macro is processed as a block, at the position of the macro in the primary POU. Commands for manipulating the order of execution therefore operate only within the macro.

'Extras' 'Expand macro'

With this command, the selected macro is re-expanded and the elements contained in it are inserted in the POU at the macro's location. The connections to the pins of the macro are again displayed as connections to the in- or outputs of the elements. If the expansion of the macro can not occur at the location of the macro box for lack of space, the macro is displaced to the right and down until enough space is available.

Note: If the project is saved under project version number 2.1, the macros will likewise all be expanded. All macros will also be expanded before conversion into other languages.

'Extras' 'Back one macro level',
'Extras' 'Back all macro level'

Symbols:

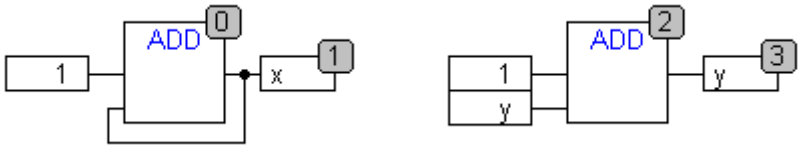
These commands are also available in the toolbar, as soon as a macro is opened for editing. If macros are nested within one another, it is possible to switch to the next higher or to the highest display level.

Feedback paths

Feedback paths can only be displayed directly in the continuous function chart editor and not in the usual function block diagram editor. Here it should be observed that the output of a block always carries an internal intermediate variable. The data type of the intermediate variable results, for operators, from the largest data type of the inputs.

The data type of a constant is obtained from the smallest possible data type, that is the constant '1' adopts the data type SINT. If now an addition with feedback and the constant '1' is executed, the first input gives the data type SINT and the second is undefined because of the feedback. Thus the intermediate variable is also of the type SINT. The value of the intermediate variable is only then allocated to the output variable.

The diagram below shows an addition with feedback and an addition with a variable. The variables x and y should be of the type INT here.



There are differences between the two additions:

The variable y can be initialised with a value which is not equal to zero but this is not the case for intermediate variable for the left addition.

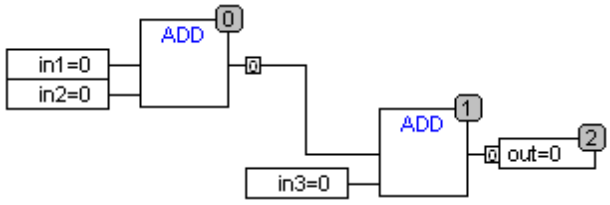
The intermediate variable for the left addition has the data type SINT while that on the right has the data type INT. The variables x and y have different values after the 129th call up. The variable x, although it is of the type INT, contains the value -127 because the intermediate variable has gone into overflow. The variable y contains the value 129, on the other hand.

CFC in Online mode

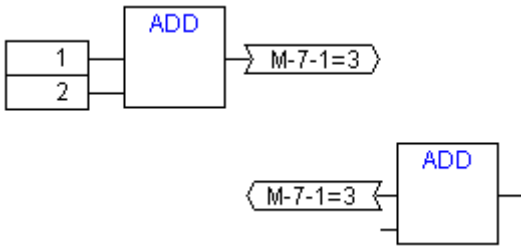
Monitoring:

The values for inputs and outputs are displayed within the input or output boxes. Constants are not monitored. For non-boolean variables, the boxes are expanded to accommodated the values displayed. For boolean connections, the variable name as well as the connection are displayed in blue if the value is TRUE, otherwise they remain black.

Internal boolean connections are also displayed Online in blue in the TRUE state, otherwise black. The value of internal non-boolean connections is displayed in a small box with rounded corners on the output pin of the connection.



PINs in macros are monitored like in- or output boxes.



Non-boolean connections with connection markers display their value within the connection marker. For boolean connections, the lines as well as the marker names are displayed in blue if the line is carrying the value TRUE, otherwise black.

Flow control:

When flow control is switched on, the connections that have been traversed are marked with the color selected in the project options.

Breakpoints:

Breakpoints can be set on all elements that also have a processing sequence order index. The processing of the program will be halted prior to execution of the respective element, that is for POUs and outputs before the assignment of inputs, for jump labels before execution of the element with the next index. The processing sequence index of the element is used as the breakpoint position in the Breakpoint dialog.

The setting of breakpoints on a selected element is accomplished with the F9 key or via the menu item 'Breakpoint on/off' in the 'Online' or 'Extras' menu or in the editor's context menu. If a breakpoint is set on an element, then this will be erased and reversed the next time the command 'Breakpoint on/off' is executed. In addition, the breakpoint on an element can be toggled by double-clicking on it.

Breakpoints are displayed in the colors entered in the project options.

RETURN label:

In Online mode, a jump label with the name „RETURN“ is automatically generated in the first column and after the last element in the editor. This label marks the end of the POU and is jumped to when stepping just before execution leaves the POU. No RETURN marks are inserted in macros.

Stepping:

When using 'Step over' the element with the next-higher order index will always be jumped to. If the current element is a macro or a POU, then its implement branches when 'Step in' is in effect. If a 'Step over' is executed from there, the element whose order index follows that of the macro is jumped to.

6 The Resources

In the **Resources** register card of the Object Organizer, there are objects for configuring and organizing your project and for keeping track of the values of the variables:

- **Global Variables** that can be utilized in the entire project; the global variables of the project as well as the libraries.
- **Library Manager** for handling all libraries which are included to the project
- **Log** for recording the activities during the online sessions
- **Parameter Manager (**formerly 'Object Dictionary') for managing variables, which are also accessible for other participants in the network (it depends on the target settings, whether this functionality is available).
- **PLC Browser** for monitoring of information from the PLC
- **PLC Configuration** for configuring your hardware
- **Sampling Trace** for graphic logging of variable values
- **Target settings** for selecting the hardware platform (target) and if available for customizing target specific parameters
- **Task Configuration** for controlling your program control via tasks
- **Watch and Receipt Manager** for indicating and presetting variable values
- **Tools** for linking external tools, which then can be started in CoDeSys. This functionality will only be available if defined in the corresponding target settings.
- Additionally there might be created and loaded a **Docuframe file** which offers a set of comments for the project variables (e.g. in a certain language), which will be printed when documenting the project with 'Project' 'Document'.

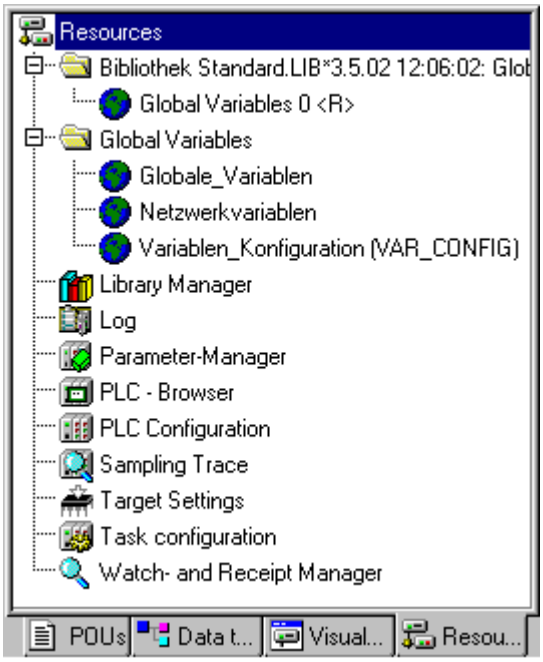


Image 6.1: Resources

6.1 Global Variables

Objects in 'Global Variables'

In the Object Organizer, you will find three objects in the **Resources** register card in the **Global Variables** folder (default names of the objects in parentheses).

- Global Variables List (Global Variables)
- Variables Configuration (Variable Configuration)

All variables defined in these objects are recognized throughout the project.

If the global variables folder is not opened up (plus sign in front of the folder), you can open it with a doubleclick <Enter> in the line.

Select the corresponding object. The **'Object Open'** command opens a window with the previously defined global variables. The editor for this works the same way as the declaration editor.

Several Variables Lists

Global variables, global network variables (**VAR_GLOBAL),** and variable configurations (**VAR_CONFIG**) must be defined in separate objects.

If you have declared a large number of global variables, and you would like to structure your global variables list better, then you can create further variables lists.

In the Object Organizer, select the **Global Variables** folder or one of the existing objects with global variables. Then execute the **'Project' 'Object Add'** command. Give the object that appears in the dialog box a corresponding name. With this name an additional object will be created with the key word **VAR_GLOBAL**. If you prefer an object a variable configuration, change the corresponding key word to **VAR_CONFIG**.

6.1.1 What are Global Variables

„Normal“ variables, constants or remanent variables that are known throughout the project can be declared as global variables, but also network variables that are also used for data exchange with other network subscribers.

With the help of **network variables**, automatic data exchange can be achieved (compare this to the non-automatic kind that uses the OD directory) within a **CoDeSys** compatible controller network that requires no controller-specific functions. For this, the network subscribers must use identical declaration lists and matching transfer configurations for network variables in their projects. In order to make this possible it is recommended that the declaration not be entered manually in each controller application, but loaded from a separate file when creating the list. (see 'Creating a global variables list').

Network variables

By an automatic data exchange (compare this to the non-automatic kind used by the Parameter Manager) it is possible to update the value of a network variable on several controller systems within a **CoDeSys** compatible controller network. This requires no controller-specific functions but the network subscribers must use

identical declaration lists and matching transfer configurations for network variables in their projects. In order to make this possible it is recommended that the declaration not be entered manually in each controller application, but loaded from a separate file when creating the list. (see below).

Create a Global Variable List

To create a Global Variable List, open the register ‘Resources’ in the Object Organizer and select the entry ‘Global Variables’ or select an already existing list. Then choose the command **‘Project‘ ‘Object‘ ‘Insert'** to open the dialog Global variable list.

This dialog can also be opened by the command **‘Project‘ ‘Object‘ ‘Properties'** which is available if an existing Global Variable List is marked in the object organizer. It shows the configuration of this list..

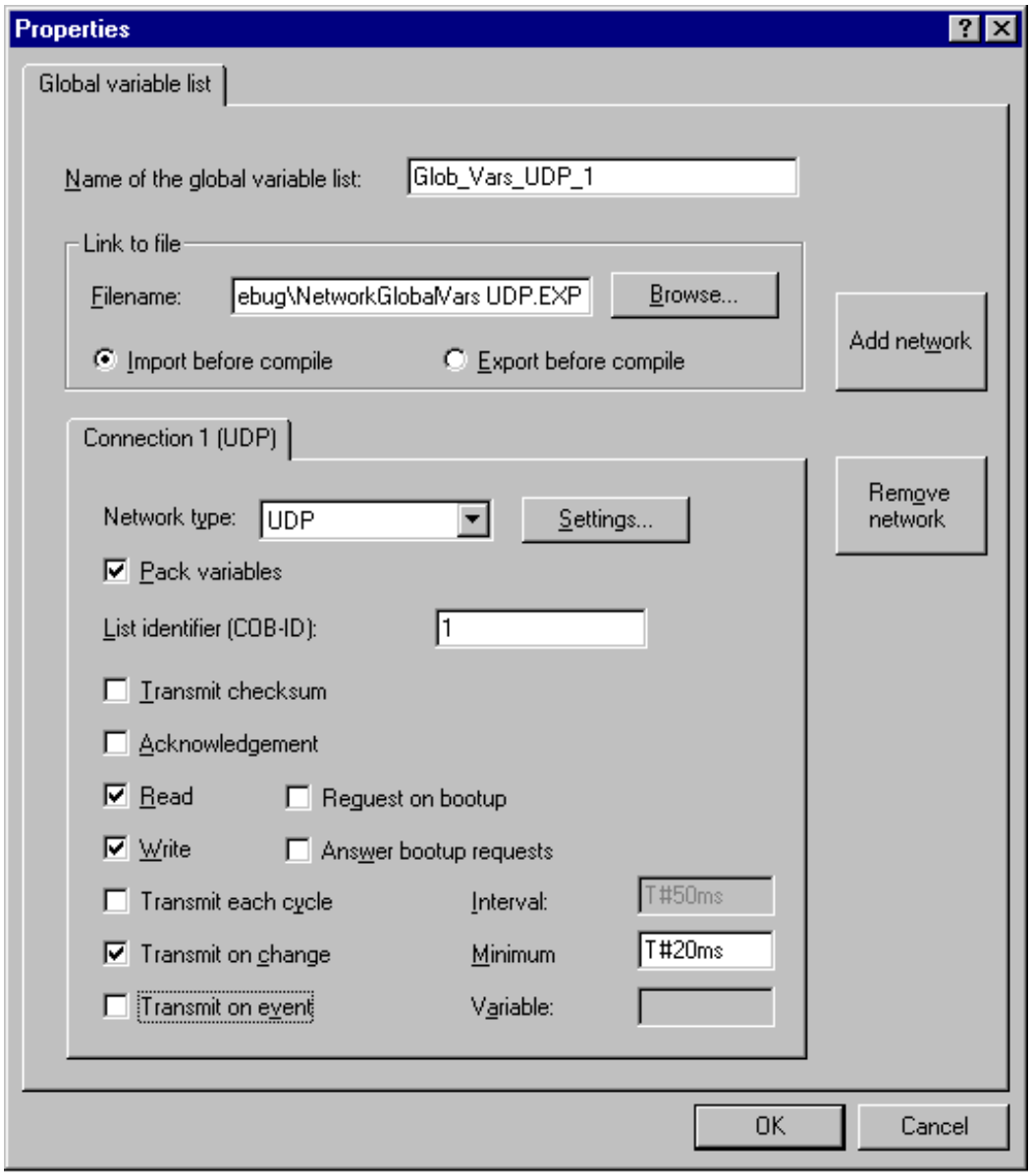


Image 6.2: Dialog to create a new Global Variable List

If the option ‘Support network variables’ is activated in the target settings, then the button <Add network> is available. Pressing this button the dialog gets extended and looks like shown in the picture. If the option is not activated, the button is not available.

Insert a **Name of the global variable list**. If you have an export file (*.exp) or a DCF file, which contains the desired variables, you can set up a link to this file (**Link to file**). To do this, insert the path of the file in the field **Filename** resp. press the button **Browse** to get the standard dialog 'Select text file'. DCF files are converted to ICE syntax when they are read in.

Activate option **Import before compile**, if you wish that the variable list will be read from the external file before each compilation of the project. Activate the option **Export before compile**, if you want the variable list to be written to the external file before each compilation of the project.

In the lower part of the dialog you can create configuration sets for up to four network connections, each referring to an own network variables list. The configuration parameters will determine how the data exchange for the particular variables list will be done. In order for the exchange in the network to work as intended, the same variable list must be compatibly configured to match in the other network subscribers.

If no configuration is yet present, you will get in the case of a CAN network a single tabulator sheet with the inscription **'Connection 1 (CAN)'**. Each time the 'Add network' button is pressed again, you get up to four more sheets inscribed with serial numbers after „Connection“.

Choose the desired type from the list following **Network type**. The list is defined by the target system entries. For example, „CAN“ as an abbreviation for a CAN network, or „UDP“ for a UDP transmission system, might be selectable.

The button **Settings** opens the dialog "Settings for <networktype>" with the following configuration parameters:

Use standard: If this button is pressed, Port 1202 will be defined for the data exchange with the other network particiapnts. The Broadcast/Multicastaddress will be set to "255 . 255 . 255 . 255" , which means, that the data exchange will be done with all participants in the network.

Port for all networks: Enter here a desired port number to overwrite the standard setting (see above, Use standard). Make sure that the other nodes in the network define the same port! If you have more than one UDP connection defined in the project then the port number will be automatically modified in all configuration sets according to the input you make here.

Broadcast/Multicast address: Enter here an address resp. the address range of a sub-network, if you want to overwrite the standard setting (e.g. "197 . 200 . 100 . 255", if you want to communicate with all nodes with IP-addresses 197 . 200 . 100 . x).

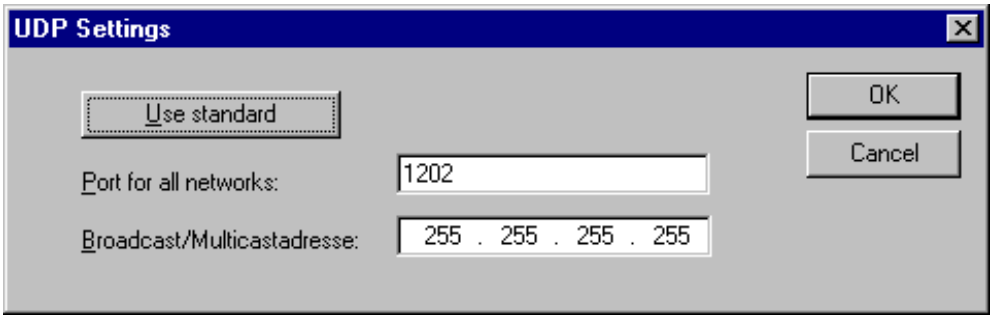


Image 6.3: Dialog 'Settings for UDP

The following options can be activated or deactivated in configuring the transmission behavior of the variables:

Pack variables: The variables are assembled for transfer into packets whose size depends on the network. If the option is deactivated, a packet is set up for each variable.

Variable telegram number: Identification number of the first packet, in which the variables will be sent. (default = 1). Further packets will be numbered in ascendent order.

Include Checksum: A checksum will be added to each packet which is sent. The checksum will be checked by the receiver to make sure that the variable definitions of sender and receiver are identic. A packet with a non-matching chechsum will not be accepted and – if this is configured ('Use acknowledge transfer', see below) – will be acknowledged negatively.

Use acknowledged transfer: Each message will be acknowledged by the receiver. As soon as the sender does not get at least one acknowledgement within a cycle, an error message will be produced.

Read: The variables in the list are read; if the option is deactivated, further variables sent over the net will be ignored.

> **Request at Bootup**: If the local node is a "reading" node (Option 'Read' activated), then as soon as it gets re-booted the actual variable values will be requested from all writing nodes and will be sent by those, independently of any other transmit conditions (time, event), which normally trigger the communication. Precondition: In the configuration of the writing nodes the option 'Answer Bootup requests' must be activated ! (see below).

Write: The variables are written; the following options apply:

> **Answer Bootup requests**: If the local node is a "writing" node (Option 'Write' activated), then each request of a reading node which is sent by it at bootup (Option Request on Bootup, see above), will be answered. That means that the actual variable values will be transmitted even if none of the other defined transmission triggers (time or event) would force this at this moment.

Transmit each cycle: Variables are written within the intervals specified after **Interval**. (time notation e.g. T#70ms).

Transmit on change: Variables are written only when their values change; an entry after Minimum can, however, set a minimum time lapse between transfers.

Transmit on event: The variables of the list will be written as soon as the variable inserted at Variable gets TRUE.

Base identifier: The packets in which the variables are sent are given an identification number which in the standard case begins with 1 and is assigned in rising order, so long as you do not enter another starting number here.

If you close the 'Global variable list' dialog with **OK**, the new object is created. Network global variables lists can be recognized in the Object Organizer by the symbol . With the command 'Project' 'Object' 'Properties' you can re-open the 'Global variable list' configuration dialog for the entry marked in the Object Organizer.

Note: If a network global variable is used on one or more **tasks**, the following applies to the time component of the transfer: When each task is called it is tested to determine which parameters apply to the transfer of the variable value (configuration in the 'Global variables list' dialog). The variable value will be transferred or not, depending on whether the specified time interval has passed. At each transfer the time interval counter for this variable is reset to zero.

Sending is always undertaken from the run-time system of the controller affected. Thus no control-specific functions have to be provided for the data exchange.

Editing Global Variable and Network Variable Lists

The editor for global variables works similar to the declaration editor. But note that you cannot edit in this editor an list, which is an image of an linked external variable list ! External variable lists only can be edited externally and they will be read at each opening and compiling of the project.

Syntax:

```
VAR_GLOBAL
    (* Variables declarations *)
END_VAR
```

Network variables are also defined in this syntax.

Example of a network variables list which was created by linking of an export file *.exp and which got the name NETWORKVARIABLES_UDP:

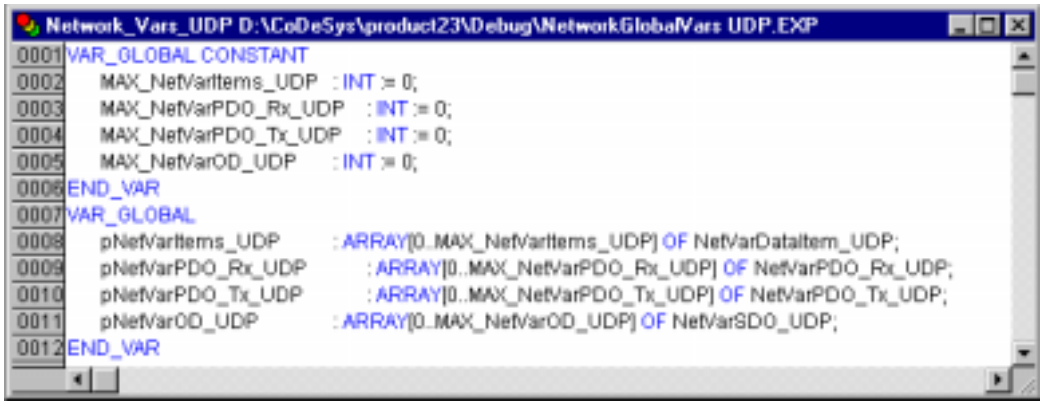

Network_Vars_UDP D:\CoDeSys\product23\Debug\NetworkGlobalVars UDP.EXP
0001 VAR_GLOBAL CONSTANT
0002 MAX_NetVarItems_UDP : INT := 0;
0003 MAX_NetVarPDO_Rx_UDP : INT := 0;
0004 MAX_NetVarPDO_Tx_UDP : INT := 0;
0005 MAX_NetVarOD_UDP : INT := 0;
0006 END_VAR
0007 VAR_GLOBAL
0008 pNetVarItems_UDP : ARRAY[0..MAX_NetVarItems_UDP] OF NetVarDataItem_UDP;
0009 pNetVarPDO_Rx_UDP : ARRAY[0..MAX_NetVarPDO_Rx_UDP] OF NetVarPDO_Rx_UDP;
0010 pNetVarPDO_Tx_UDP : ARRAY[0..MAX_NetVarPDO_Tx_UDP] OF NetVarPDO_Tx_UDP;
0011 pNetVarOD_UDP : ARRAY[0..MAX_NetVarOD_UDP] OF NetVarSDO_UDP;
0012 END_VAR

Image 6.4: Example of a network variables list

Editing the lists for remanent Global Variables

If they are supported by the runtime system, remanent variables may be processed. There are two types of remanent global variables:

Retain variables remain unchanged after an uncontrolled shutdown of the runtime system (off/on) or an 'Online' 'Reset' in **CoDeSys**. **Persistent variables** remain unchanged after a controlled shutdown of the runtime system (stop, start) or an 'Online' 'Cold reset' or a download.

Persistent variables are not automatically also Retain variables !

Remanent variables are additionally assigned the keyword **RETAIN** or **PERSISTENT**.

Network variables are also defined in this syntax.

Syntax:

```
VAR_GLOBAL RETAIN
    (* Variables declarations *)
END_VAR

VAR_GLOBAL PERSISTENT
    (* Variables declarations *)
END_VAR
```

Global Constants

Global constants additionally get the keyword **CONSTANT.**

Syntax:

```
VAR_GLOBAL CONSTANT
    (* Variables declarations *)
END_VAR
```

6.1.2 Variable Configuration

In function blocks it is possible to specify addresses for inputs and outputs that are not completely defined, if you put the variable definitions between the key words **VAR** and **END_VAR**. Addresses not completely defined are identified with an asterisk.

Example:

```
FUNCTION_BLOCK locio
VAR
   loci AT %I*: BOOL := TRUE;
   loco AT %Q*: BOOL;
END_VAR
```

Here two local I/O-variables are defined, a local-In (%I*) and a local-Out (%Q*).

If you want to configure local I/Os for variables configuration in the Object Organizer in the **Resources** register card, the object **Variable_Configuration** will generally be available. The object then can be renamed and other objects can be created for the variables configuration.

The editor for variables configuration works like the declaration editor.

Variables for local I/O-configurations must be located between the key words **VAR_CONFIG** and **END_VAR**.

The name of such a variable consists of a complete instance path through which the individual POUs and instance names are separated from one another by periods. The declaration must contain an address whose class (input/output) corresponds to that of the incompletely specified address (%I*, %Q*) in the function block. Also the data type must agree with the declaration in the function block.

Configuration variables, whose instance path is invalid because the instance does not exist, are also denoted as errors. On the other hand, an error is also reported if no configuration exists for an instance variable. In order to receive a list of all necessary configuration variables, the **'All Instance Paths'** menu item in the 'Insert' menu can be used.

Example:
Assume that the following definition for a function block is given in a program:

```
PROGRAM PLC_PRG
VAR
   Hugo: locio;
   Otto: locio;
END_VAR
```

Then a corrected variable configuration would look this way:

```
VAR_CONFIG
   PLC_PRG. Hugo.loci AT %IX1.0 : BOOL;
   PLC_PRG. Hugo.loco AT %QX0.0 : BOOL;
   PLC_PRG. Otto.loci AT %IX1.0 : BOOL;
   PLC_PRG.Otto.loco AT %QX0.3 : BOOL;
END_VAR
```

Note: Be aware not to describe an output, which is used in the variables configuration, additionally within the project or by a variable (AT declaration). This would not be noticed.

‚Insert' 'All Instance Paths'

With this command a **VAR_CONFIG - END_VAR**-block is generated that contains all of the instance paths available in the project. Declarations already on hand do not need to be reinserted in order to contain addresses already in existence. This menu item can be found in the window for configuration of variables if the project is compiled ('**Project**' '**Rebuild All**').

6.1.3 Document Frame

If a project is to receive multiple documentations, perhaps with German and English comments, or if you want to document several similar projects that use the same variable names, then you can save yourself a lot of work by creating a docuframe with the **'Extras' 'Make Docuframe File'** command.

The created file can be loaded into a desired text editor and can be edited. The file begins with the **DOCUFILE** line. Then a listing of the project variables follows in an arrangement that assigns three lines to each variable: a **VAR** line that shows when a new variable comes; next, a line with the name of the variable; and, finally, an empty line. You can now replace this line by using a comment to the variable. You can simply delete any variables that you are unable to document. If you want, you can create several document frames for your project.

In order to use a document frame, give the **'Extras' 'Link Docu File'** command. Now if you document the entire project, or print parts of your project, then in the program text, there will be an insertion of the comment produced in the docuframe into all of the variables. This comment only appears in the printout!

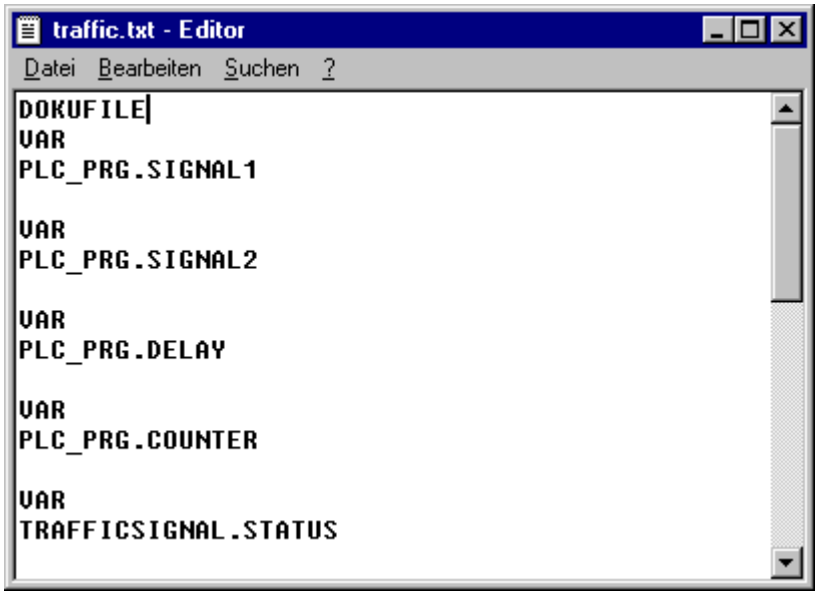


Image 6.5: Windows Editor with Document Frame

'Extras' 'Make Docuframe File'

Use this command to create a document frame. The command is at your disposal, whenever you select an object from the global variables.

A dialog box will open for saving files under a new name. In the field for the **name file**, the *.txt extension has already been entered. Select a desired name. Now a text file has been created in which all the variables of your project are listed.

'Extras' 'Link Docu File'

With this command you can select a document frame.

The dialog box for opening files is opened. Choose the desired document frame and press **OK**. Now if you document the entire project, or print parts of your project, then in the program text there will be an insertion of the comment produced in the docuframe into all of the variables. This comment only appears in the printout!

To create a document frame, use the **'Extras' 'Make Docuframe File'** command.

Library Manager

The library manager shows all libraries that are connected with the current project. The POUs, data types, and global variables of the libraries can be used the same way as user-defined POUs, data types, and global variables.

The library manager is opened with the **'Window' 'Library Manager'** command. Information concerning included libraries is stored with the project and can be viewed in the dialog 'Informations about external library'. To open this dialog select the corresponding library name in the library manager and execute the command 'Extras' 'Properties'.

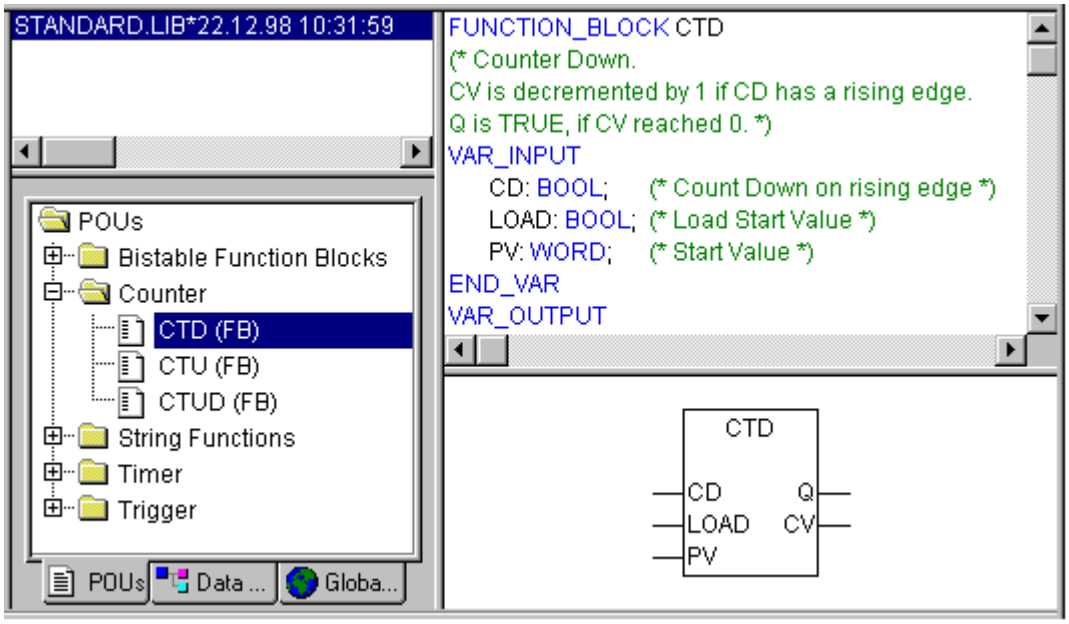


Image 6.1: Library Manager

Using the Library Manager

The window of the library manager is divided into three or four areas by screen dividers. The libraries attached to the project are listed in the upper left area.

In the area below that, depending on which register card has been selected, there is a listing of the **POUs , Data types, Visualizations** or **Global variables** of the library selected in the upper area.

Folders are opened and closed by doubleclicking the line or pressing <Enter>. There is a plus sign in front of closed folders, and a minus sign in front of opened folders.

If a POU is selected by clicking the mouse or selecting with the arrow keys then the declaration of the POU will appear in the upper right area of the library manager; and in the lower right is the graphic display in the form of a black box with inputs and outputs.

With data types and global variables, the declaration is displayed in the right area of the library manager.

Standard Library

The library with "standard.lib" is always available. It contains all the functions and function blocks which are required from the IEC61131-3 as standard POUs for an IEC programming system. The difference between a standard function and an operator is that the operator is implicitly recognized by the programming system, while the standard POUs must be tied to the project (standard.lib).

The code for these POUs exists as a C-library and is a component of **CoDeSys**.

User-defined Libraries

If a project is to be compiled in its entity and without errors, then it can be saved in a library with the **'Save as'** command in the **'File'** menu. The project itself will remain unchanged. An additional file will be generated, which has the default extension ".lib". This library afterwards can be used and accessed like e.g. the standard library.

For the purpose to have available the POUs of a project in other projects, save the project as an **Internal Library *.lib**. This library afterwards can be inserted in other projects using the library manager.

If you have implemented POUs in other programming languages, e.g. C, and want to get them into a library, then save the project using data type **External Library *.lib)**. You will get the library file but additonally a file with the extension ”*.h”. This file is structured like a C header file and contains the declarations of all POUs, data types and global variables, which are available with the library. If an external library is used in a project, then in simulation mode that implementation of the POUs will be executed, which was written with CoDeSys; but on the target the C-written implementation will be processed.

If you want to add licensing information to a library, then press button **Edit license info...** and insert the appropriate settings in the dialog 'Edit Licensing Informationen'. See the description at 'File' 'Save as...' resp. the documentation on the 'License Management in CoDeSys' (Chapter 8).

'Insert' 'Additional Library'

With this command you can attach an additional library to your project.

In the dialog box for opening a file, choose the desired library with the "*.lib" extension. The library is now listed in the library manager, and you can use the objects in the library as user-defined objects.

As soon as you include a library for which a license (see Chapter 8, 'License Management in CoDeSys'.) is needed and no valid license is found, you may get a message that the library is only available in demo mode or that the library is not licensed for the currently set target. You can ignore this message at that time or start appropriate actions concerning the license. An invalid license will produce an error during compile ('Project' 'Build'). In this case a doubleclick on the error message resp. <F4> will open the dialog 'License information' where you can start the appropriate actions.

Remove Library

With the **'Edit' 'Delete'** command you can remove a library from a project and from the library manager.

'Extras' 'Properties'

This command will open the dialog 'Informations about internal (resp. external) library'. For internal libraries you will find there all data, which have been inserted in the Project Info when the library had been created in CoDeSys. For external libraries the library name and library path will be displayed.

6.3 Log

The log stores in chronological order actions that occur during an Online session. For this purpose a binary log file (*.log) is set up. Afterward, the user can store excerpts from the appropriate project log in an external log.

The log window can be opened in either Offline or Online mode and can thus serve as a direct monitor online.

'Window' 'Log'

To open, select the menu item 'Window' 'Log' or select entry 'Log' in the Resources tab.

In the log window, the filename of the currently displayed log appears after **Log:**. If this is the log of the current project, the word "(Internal)" will be displayed.

Registered entries are displayed in the log window. The newest entry always appears at the bottom.

Only actions belonging to categories that have been activated in the 'Filter' field of the menu 'Project' 'Options' 'Log' will be displayed.

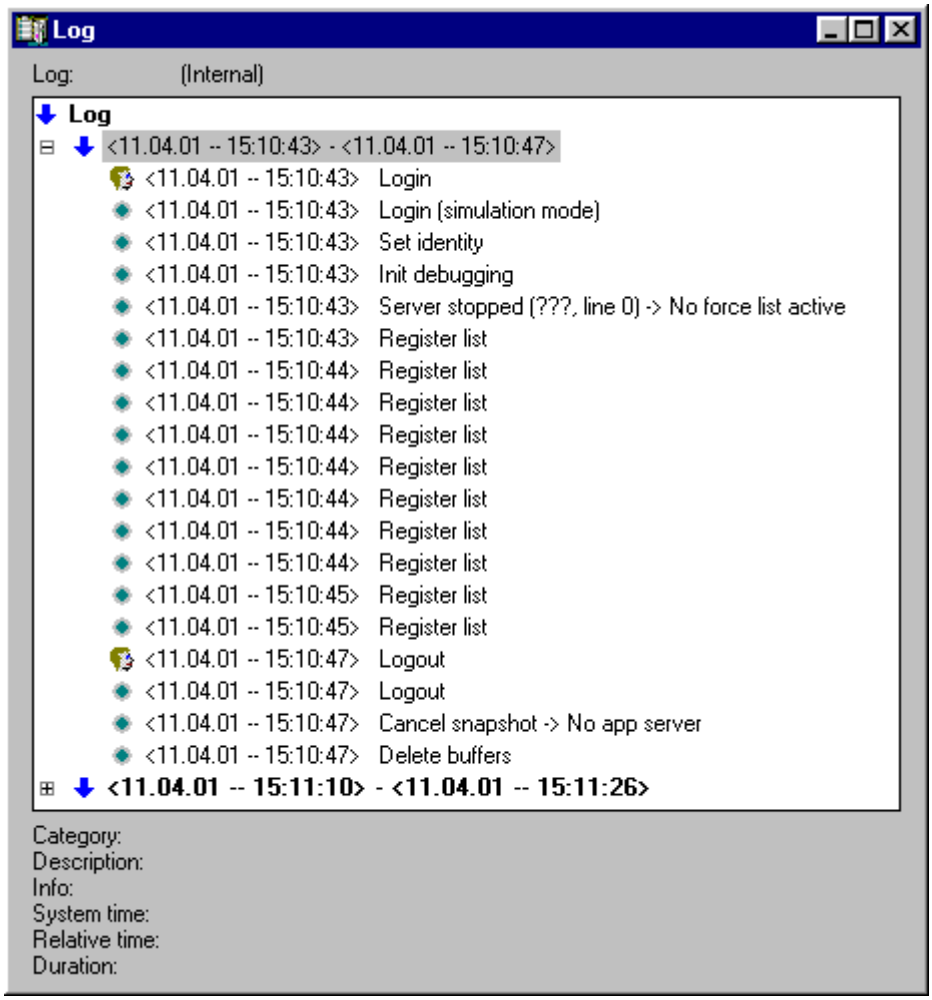


Bild 6.2:Log window

Available information concerning the currently selected entry is displayed below the log window:

Category: The category to which the particular log entry belongs. The following four categories are possible:

- User action: The user has carried out an Online action (typically from the Online menu).
- Internal action: An internal action has been executed in the Online layer (e.g. Delete Buffers or Init Debugging).
- Status change: The status of the runtime system has changed (e.g. from Running to Break, if a breakpoint is reached).
- Exception: An exception has occurred, e.g. a communication error.

Description: The type of action. User actions have the same names as their corresponding menu commands; all other actions are in English and have the same name as the corresponding `OnlineXXX()` function.

Info: This field contains a description of an error that may have occurred during an action. The field is empty if no error has occurred.

System time: The system time at which the action began, to the nearest second.

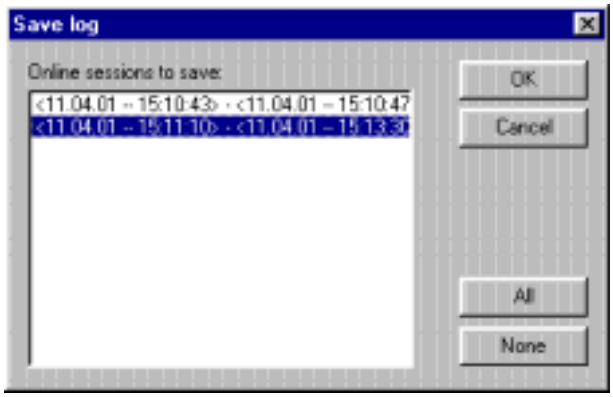
Relative time: The time measured from the beginning of the Online session, to the nearest millisecond.

Duration: Duration of the action in milliseconds.

Menu Log

When the log window has the input focus, the menu option **Log** appears in the menu bar instead of the items 'Extras' and 'Options'.

The menu includes the following items:

Load…	An external log file *.log can be loaded and displayed using the standard file open dialog. The log that is present in the project will not be overwritten by the command. If the log window is closed and later opened again, or a new Online session is started then the version that is loaded will again be replaced by the project log.
Save…	This menu item can only be selected if the project log is currently displayed. It allows an excerpt of the project log to be stored in an external file. For that, the following dialog will be displayed, in which the Online sessions to be stored can be selected: After successful selection, the standard dialog for storing a file opens ('Save Log'). 
Display Project Log	This command can only be selected if an external log is currently displayed. It switches the display back to the project log.

Storing the project log

Regardless of whether or not the log is stored in an external file (see above), the project log is automatically stored in a binary file entitled <projectname>.log. If a different path is not explicitly given in the 'Project' 'Options' 'Log' dialog, the file is stored in the same directory as that in which the project is stored.

The maximum number of Online sessions to be stored can be entered in the 'Project' 'Options' 'Log' dialog. If this number is exceeded during recording, the oldest session is deleted to make room for the newest one.

6.4 PLC Browser

The PLC-browser is a text-based control monitor (terminal). Commands for the request of specific information from the controller are entered in an entry line and sent as string to the controller. The returned response string is displayed in a result window of the browser. This functionality serves diagnostic- and debugging purposes.

The commands available for the set target-system are made up of the **CoDeSys** standard set plus a possible extension set of the controller manufacturer. They are managed in an ini file and implemented accordingly in the runtime system.

General remarks concerning PLC Browser operation

Select the entry PLC-Browser in the Resources tab-control. (availability depends upon the target-system settings)

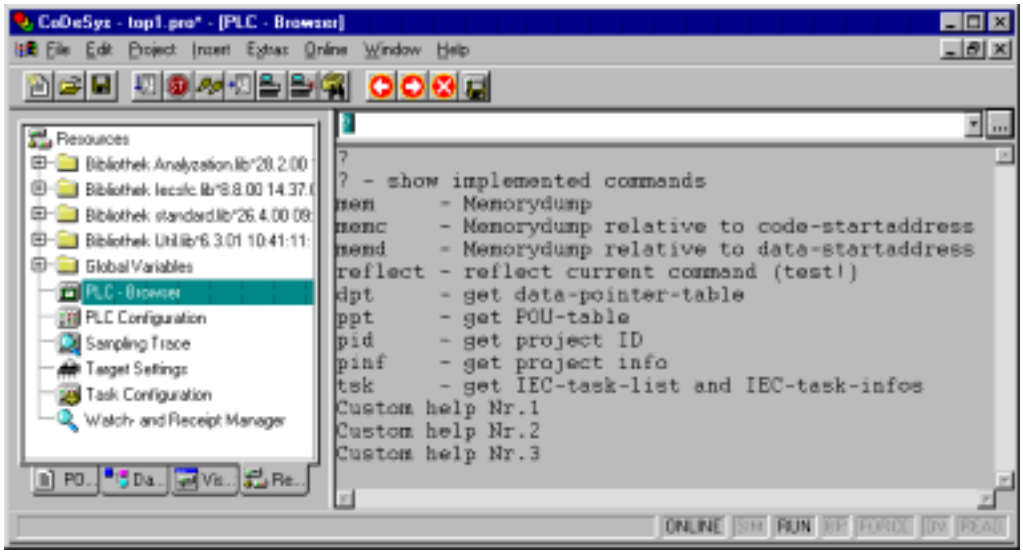


*Image 6.3: The **CoDeSys** PLC Browser*

The browser consists of a command entry line and a result/display window.

In a selection box the input line displays a list of all the commands entered since the start of the project (input history). They are available for re-selection until the project is closed. Only commands, which differ from those already existing, are added to the list.

The entered command is sent to the controller with <Enter>. If there is no Online connection, the command is displayed in the result window in the same way as it is sent to the controller, otherwise the response from the controller is shown there. If a new command is sent to the controller, the content of the result window is deleted.

Commands can be entered in the form of command strings (see below, command entry), the use of macros is possible as well (see below, use of macros):

Command entry in the PLC Browser

Basically the PLC-Browser makes available the **3S standard commands** hard-coded in the run-time system. It is concerned with functions for direct memory manipulation, for the output of project- and status functions as well as for run-time monitoring. They are described in the browser's **ini-file**, which is an integral part of the Target Support Package. These standard commands can be further supplemented by specialized ones, e.g. self-diagnostic functions or other status messages of the control application. The expansion of the command list must be carried out both in the customer interface in the run-time system as well as through additional entries in the Browser ini-file.

When opening the project the **command list** available in the PLC-Browser is generated from the entries in the Browser ini-file. It can be accessed as input help using the **...** key in the „Insert standard command" or using <F2>. The command can be typed in or selected from the list with a double-click.

The general command syntax is:

<KEYWORD><LEER><KEYWORD-DEPENDEND PARAMETERS>

The keyword is the **command**. With which **parameter**s it can be expanded is described in the respective tooltip in the entry help window.

The command, which has been sent, is repeated in the output data window, the controller's response appears below it.

Example: Request for the project Id from the controller with the command "pid"

Entry in command line:

```
pid.....
```

Output in result window:

```
pid
Project-ID:  16#0025CFDA
```

A **help text** can be supplied for each standard command with ?<BLANK><KEYWORD>. This is similarly defined in the ini-file.

The following commands are firmly integrated in the run-time system and contained in the ini-file with the corresponding entries for entry help, tooltips and help.

Command	Description
?	The run-time system supplies a list of the available commands
mem	Hexdump of a memory area
memc	Hexdump relative to the start address of the code in the controller
memd	Hexdump relative to the database address in the controller
reflect	Reflect current command line, for test purposes
dpt	Read data-pointer table
ppt	Read POU table

pid	Read project Id
pinf	Read project info
tsk	Show IEC-task list containing task infos.
od	Edit Object Dictionnary
pod	Edit Definition of network variables
startprg	Start PLC program
stopprg	Stop PLC program
resetprg	Reset PLC program
resetprgcold	Reset PLC program cold
resetprgorg	Reset PLC program original
reload	Reload boot project
getprgprop	Program properties
getprgstat	Program status
filecopy	Copy file [from] [to]
filerename	Rename files [old] [new]
filedelete	Delete file [filename]

Note: The first word of the command sequence entered is interpreted as keyword. If „<SPACE>“ follows a keyword (e.g. „mem ?“), the ini-file is searched for the existence of a help section to this keyword. If one is available, nothing is sent to the controller, but only the help text is displayed in the output data window.

If the first word of the command entry (<KEYWORD>) is not recognized by the controller, the response ‘Keyword not found’ appears in the result window.

Use of macros during the command entry in PLC-Browser

If a command associated with a macro is entered in the command line, this is expanded before it is sent to the controller. Then the response in the result window appears in a similarly expanded form.

The entry syntax is: <KEYWORD><macro>

<KEYWORD> is the command.

Macros are:

%P<NAME>	If NAME is a POU-name, the expression is expanded to <POU-Index>, otherwise there is no alteration
%V<NAME>	If NAME is a variable name, the expression is expanded to #<INDEX>:<OFFSET>, otherwise there is no alteration (this notation #<INDEX>:<OFFSET> is interpreted by the controller as a memory address)
%T<NAME>	If NAME is a variable name, the expression is expanded to

	<VARIABLENTYP>, otherwise there is no alteration.
<%S><NAME>	If NAME is a variable name, the expression is expanded to <SIZEOF(VAR)>, otherwise there is no alteration.

The % character is ignored if the escape symbol \ (Backslash) is placed in front. The escape symbol as such is only transmitted if written \\.

Example:
Entry in command line: (memory dump of the variable .testit ?)

```
mem %V.testit
```

Output in result window:

```
mem #4:52
03BAAA24  00  00  00  00  CD  CD  CD  CD   ....ÍÍÍÍ
```

Further PLC Browser options

In the '**Extras**' menu or in the PLC-Browser's toolbar there are the following commands for handling the command entry or history list:

With **History forward** and **History backward** you can scroll backwards and forwards through the query results already carried out. The history recording is continued until you leave the project.

With **Cancel command** you can break off a query which has been initiated.

With **Save history list** you can save the query results carried out up until that point in an external text file. The dialogue 'Save file as' will appear, in which you can enter a file name with the extension „.bhl" (Browser History List). The command **Print last command** opens the standard dialogue to print. The current query plus the output data in the message window can be printed.

6.5 Parameter Manager

PLEASE NOTE: **The 'Parameter Manager' will replace the 'Object Dictionary' in the future. Currently still the Object Dictionary functionality is implemented as supplied before, but the entry in the Resources tab of the Object Organizer has been renamed already.**

So the following description has been left unchanged and will still use the name 'Object Dictionary'.

Apart from the use of network variables (see Chapter 4.1.1), the Object Dictionary is also available for use for the data exchange, between different **CoDeSys**-compatible controllers within a network, which does not require any controller specific functions.

In the description which follows "OD" stands for Object Dictionary.

Note: In order that the OD-Editor is available in the Resources, the option ‘Support Object Dictionary’ must be activated in the Target-system settings.

An OD is a data dictionary, **which can be stored in a controller together with the control program**. It makes available **data objects** of any kind and size and variable values or parameters, which are referenced unambiguously with a **Index/Sub-index**.

Thus both the local control program and other network participants / control programs can access using communication services (library functions obj_dict.lib) on variables or constants, without knowing their addresses and without having to provide their own functions for the data exchange.

Since the data are filed independently of the control program, an OD can be used to save ‘receipts’, which are also retained during the change of program. In addition a running controller can be ‘fed’ with different receipts, without having to load the program again.

OD entries can be created, processed, saved with the **CoDeSys OD Editor** and loaded into the controller (see Chapter 4.5.1). In addition an OD parameter can be reloaded from the controller into the Editor (not possible but also not necessary for variable ODs).

An entry in the OD corresponds to a line in a table, which is displayed in the OD editor. The columns in the table correspond with the fields of the entries. An entry in an OD consists of the following fields (explanations see Chapter 4.5.1):

for parameter entries:

Index	Subindex	Description	Attribute	Type	Access	Value

or for variable entries:

Index	Subindex	Attribute	Access	Variable

All the variable ODs contained in the editor are always downloaded together into the controller. Parameter ODs, on the other hand, must be selected and loaded individually.

Thus, for example, two or more identical parameter ODs can be generated in the indexing, the objects of which, however, receive different values. As required a specific record can be loaded into the controller using the corresponding OD, which is then available again to the control program, without it having to be „touched“ itself.

Example:

see marked area of data in the ConstA and ConstB ODs.

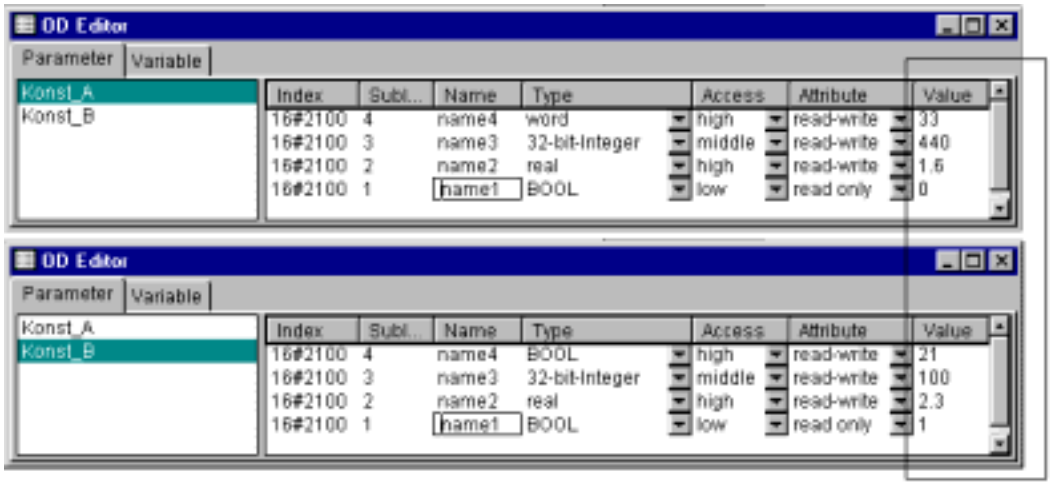


Image 6.4: OD, use of two parameter sets

The OD Editor (Parameter Manager)

The entry ‘**Parameter Manager'** can be found in the Resources tab. It opens the OD editor, with which ODs can be created, processed, saved and loaded. An OD can be loaded each time to/from the controller, to which an Online connection exists. If there is no Online connection, the OD can be created locally and saved with the project.

The Editor window consists of two halves. The left half is used for navigation, i.e. it shows the ODs loaded into the Editor. The right half of the Editor contains a table editor, the columns of which contain the field descriptions of an OD entry as predefined headings (see above).

Parameter or **Variable** entries are made on separate tabs:

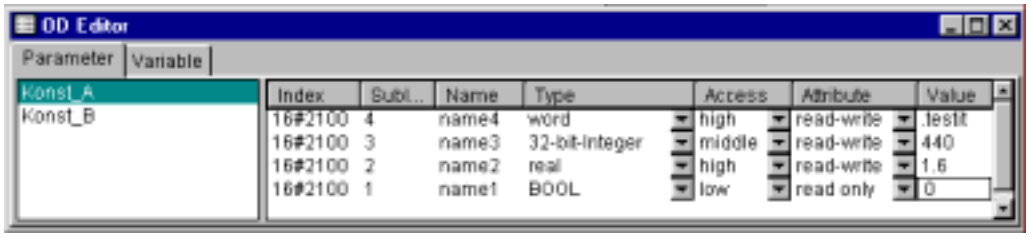


Image 6.5: Table for parameter ODs

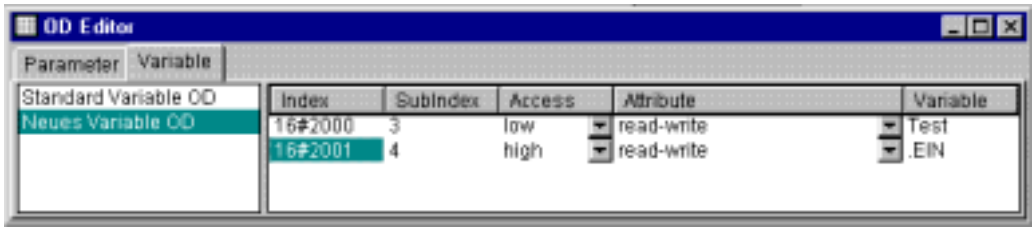


Image 6.6: Table for variable ODs

Processing/creation of an OD

The 'OD editor' menu has commands available offline for processing in the editor. These can also be found in the context menu reached via the right mouse button.

If no OD is yet saved with the current project, an empty OD appears in the Editor with the default title **Standard Parameter OD** or **Standard Variable OD** in the navigation window. You can rename this title using the **'Rename OD**' command and then fill a line in the right part of the window with values for each object entry. To enter values, click on each corresponding point of the line or column so that an editor field opens. Input assistance (<F2>) can be used. (For operation over the keyboard see also below).

Pressing the <**Enter**> key ends the input of values in the current editor field. You can then jump to another field using the arrow keys.

Using the command **'Format Dec/Hex'** you can switch the entry format between decimal and hex.

You can obtain new lines with the command **'Insert lines'**. The new line will be inserted above that in which the cursor is resting. If the new line is to be inserted below, use the key combination **<Ctrl> <Enter>**.

The line in which the cursor is resting can also be deleted with **'Delete line'**. If you want to delete several lines, you can mark them for this purpose by dragging the desired area with the left mouse button held down. The lines in the marked area receive a colored background. You can also reach this by navigating with the arrow keys with the **<Shift>** key held down.

Marked lines can be copied and pasted using the 'Copy' and 'Paste' commands in the 'Edit' menu. In this way you can also mark the entire contents of an OD and copy it into another OD.

With the commands **'Add OD'** or **'Delete OD'** you can respectively add or delete ODs.

If a new, empty OD entry line is created, the fields are automatically pre-assigned. This assumes that the user would wish to enter a list of similar values under one index. That is to say that in the Index field the same value appears as for the last entry, but the next free value is assigned for the sub-index. The remaining fields of each new entry will be filled with the default settings.

OD field entries for parameters and variables:

Index	positive integers: possible values dependent upon target-system settings (‘Network functions’, ‘Index entries’)
SubIndex:	positive integers: possible values dependent upon

	target-system settings (‘Network functions’, ‘Index entries’)
Description (only with parameter OD)	random string of a maximum of 20 characters (optional)
Type (only with parameter OD)	8/16/32-bit-Integer | byte | word | double word | real | string | long real (64bit)
Access	high | middle | low it depends on the hardware how these levels are used
Attribute	read only | write only | read-write
Value (only with parameter OD)	random value corresponding with the type entered
Variable (only with variable OD)	project variable name (using <F2> input help): for global variables a dot must be inserted before the variable name (e.g.“.var1“)

Operation over the keyboard

The Editor can be controlled solely from the keyboard. With the <**F6**> key the focus can be moved between the „Tab Controls“ and the two halves of the window. According to the focus, using the arrow keys one can switch between the tabs or between the ODs in the navigation window or input fields in the editor window.

Handling of input errors

If there are syntactical errors in the input, e.g. input of non-numerical characters in the Index/Sub-index fields, an error message appears when an attempt is made to leave the field.

If entries are made, which are not compatible with the target-system settings, during compilation error messages will appear in the message window (e.g. incorrect index range). (e.g. incorrect index range). Other input errors cannot be recognized until downloading of the OD, e.g. if the name of a variable for the download time no longer exists, because it was deleted after the creation of the OD entry or if the OD was opened in a project, in which this variable does not exist.

Entries, in which fields, which can't be empty (as, for example Index/Sub-index), are empty, will also be queried with an error message before being downloaded.

Errors, which are noted during an attempt to download, appear collated in the dialogue box **‘OD-Download: Result’**. This can be opened using **‘Extras’ ‘Last Download result’**. The erroneous entry will be tagged together with a code number describing the type of error.

List of the most important error codes

Code	Error
0	no error
1	Type mismatch (the data type delivered does not match what was expected)
2	Entry (not) existing (Entry does not exist)
3	Index not in the permitted range, entry is not created

4	Sub-index not in the permitted range
5	too little memory, in order to carry out the operation
6	Attribute or access details are not compatible with the desired operation

OD-transfer between Editor and controller

Once the processing in the Editor is completed, the ODs can be loaded into the controller in the Online mode. In so doing the whole variable OD is loaded with each download of a project, the parameter ODs, on the other hand, must be selected separately. Parameter ODs can, however, be loaded into the Editor from the controller.

For operating in Online mode, use the 'Extras' menu with the following commands:

Load current parameter OD	The current (marked) parameter OD is loaded into the controller
Load variable ODs	All variable ODs are loaded together into the controller
Last download result	The dialogue ‘OD-Download: Result:' is opened. It contains a list of the responses of the controller to the last download. Error messages are provided with a code number (identification see below)
Delete all OD entries online	All OD entries in the controller are deleted
Parameter Upload	The parameter entries of the Object Dictionary are read into the controller and displayed as a new parameter OD in the Editor with the name ‘Uploaded’.
Format Dec/Hex	Switching the display format between decimal and hex.

6.5.2 Program access to the OD (Object Dictionary, Para,eter Manager) data

With the aid of library functions a control program can access the entries of the local or the Object Dictionary (OD) of another controller. In this way individual OD entries can be read/added to/overwritten or deleted from the program, as far as its access attributes permit.

The necessary functions and function blocks are contained in the library ObjDict.lib.

6.6 PLC Configuration

The PLC Configuration is found as an object in the register card **Resources** in the Object Organizer. With the PLC Configuration editor you must describe the hardware the opened project is established for. For the program implementation the number and position of the inputs and outputs is especially important. With this

description **CoDeSys** verifies whether the IEC addresses used in the program also actually exist in the hardware.

The base of working in the configuration editor is/are the **configuration files** (*.cfg, see below **Note concerning version compatibility**') and the **device files** (.e.g. *.gsd, *.eds). These are stored in the directory which is defined in the target file (see Chapter 4.10, Target Settings) and are read when the project is opened in CoDeSys. The configuration file describes a basic configuration, which is mapped in the configuration editor and it defines to which extent the user can customize this configuration in the editor.

Attention: As soon as the underlying configuration file (*.cfg) has been modified, you have to redo the configuration in CoDeSys !

Note concerning version compatibility: In CoDeSys V2.2 a new format was implemented for the PLC Configuration. From that version on the basic configuration files have to use the extension ***.cfg**. In contrast the configuration editor in former CoDeSys versions needed configuration files with an extension ***.con**. But: In the target file you can determine that the "old" configurator should be used furtheron, even when an old project is opened in V2.2 or higher. This avoids the necessarity to create new configuration files, the *.con-files can be used furtheron. If this option is not set in the target file, then you can convert the old PLC Configuration, which is stored in the project, to the new format, if (!) an appropriate new *.cfg-file has been provided (see command 'Extras' 'Convert').

The CoDeSys configuration editor allows to configure I/O modules as well as CAN and profibus modules. After the final customization by the user a binary image of the configuration is sent to the PLC:

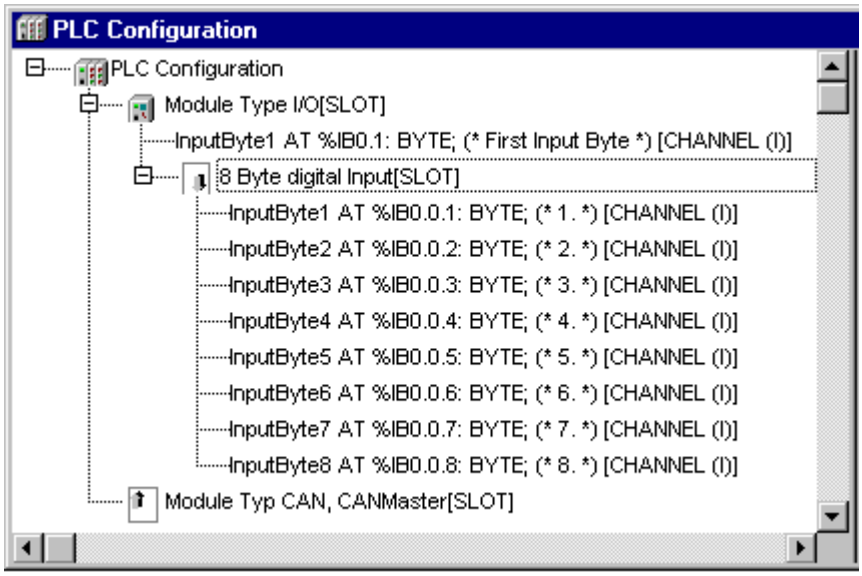

PLC Configuration
PLC Configuration
Module Type I/O[SLOT]
InputByte1 AT %IB0.1: BYTE; (* First Input Byte *) [CHANNEL (I)]
8 Byte digital Input[SLOT]
InputByte1 AT %IB0.0.1: BYTE; (* 1. *) [CHANNEL (I)]
InputByte2 AT %IB0.0.2: BYTE; (* 2. *) [CHANNEL (I)]
InputByte3 AT %IB0.0.3: BYTE; (* 3. *) [CHANNEL (I)]
InputByte4 AT %IB0.0.4: BYTE; (* 4. *) [CHANNEL (I)]
InputByte5 AT %IB0.0.5: BYTE; (* 5. *) [CHANNEL (I)]
InputByte6 AT %IB0.0.6: BYTE; (* 6. *) [CHANNEL (I)]
InputByte7 AT %IB0.0.7: BYTE; (* 7. *) [CHANNEL (I)]
InputByte8 AT %IB0.0.8: BYTE; (* 8. *) [CHANNEL (I)]
Module Typ CAN, CANMaster[SLOT]

Image 6.7: PLC Configuration with a CPU Module and a CAN Module.

The PLC Configuration is displayed in the editor in tree structure and can be edited using menu commands and dialogs. The configuration contains input and/or output elements and also management elements which themselves also have subelements (for example, a CAN-bus or a digital input card with 8 inputs).

For inputs and outputs, symbolic names can be assigned. The IEC-address where this input or output can be accessed is then located behind the symbolic name.

6.6.1 Working in the PLC Configuration

The configuration editor is divided up in two parts. In the left window the configuration tree is displayed. Structure and components of the tree result primarily (Standardconfiguration) from the definitions found in the configuration file, but can be modified by the additional adaptation which is done by the user in the CoDeSys PLC Configurator. In the right window the currently available dialog is shown.

On top of the configuration tree there is the entry of the “root” module, marked with the symbol and a name, which has been defined in the configuration file. Below you find hierarchically idented the other elements of the configuration: Modules of different types (CAN, Profibus, I/O), channels or bit channels.Selecting of elements

- For selecting elements, click the mouse on the corresponding element, or, using the arrow keys, move the dotted rectangle onto the desired element.

- Elements that begin with a plus sign are organization elements and contain subelements. To open an element, select the element and doubleclick the plus sign or press <Enter>. You can close opened elements (minus sign in front of the element) the same way.

Insert elements, ‘Insert’ ‘Insert element’, ‘Insert’ ‘Append subelement’

Depending on the definitions in the configuration file(s) and on the available device files, which have been read when the project was opened, a basic composition of elements is automatically positioned in the configuration tree. If one of those elements is selected, further elements may be added if this is allowed by the definitions in the configuration file and if the needed device files are available:

- **‘Insert’ ‘Insert element’**: An element can be selected and inserted before the element which is currently marked in the configuration tree.
- **‘Insert’ ‘Append subelement’**: An element can be selected and inserted as subelement of the element which is currently marked in the configuration tree. It will be inserted at the last position.
- The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

Replacing/switching Elements, ‘Extras’ ‘Replace element’’

Depending on the definition in the configuration file, the currently selected element may be get replaced by an other one. The same way it may be possible to switch

channels, which are set up in a way that they can be used as input or as output elements.

Symbolic names

Symbolic names for modules and channels can be defined in the configuration file. In this case they will be shown in the configuration editor before the ‘AT' of the IEC address of the respective element. In the configuration file also is defined whether the symbolic name can be edited or inserted in the configuration editor. To enter a symbolic name, select the desired module or channel in the configuration tree and open a text field by a mouseclick on the 'AT' before the IEC address. In the same manner you can edit an existing symbolic name after a doubleclick on the name. Please regard that allocating a symbolic name corresponds with a valid variable declaration !

Recalculation of Module addresses, ‘Extras’ ‘Calculate addresses’

If the option “Calculate addresses” is activated in the dialog ‘Settings’ of the PLC configuration editor (see Chapter 4.6.2), then the command ‘Extras’ ‘Compute addresses’ will start to recalculate the addresses of the modules. All modules starting with the one, which is currently selected in the configuration tree, will be regarded.

Return to standard configuration, 'Extras' 'Standard configuration'

The command 'Extras' 'Standardkonfiguration' can be used to restore the original PLC configuration, which is defined by the configuration file and saved in the project.

Converting of old PLC configurations, 'Extras' 'Convert'

This command is available in the menu 'Extras' if you open a project containing a PLC configuration, which was created with an older CoDeSys version than V2.2. If all needed configuration files are available, the command 'Convert' will transfer the existing configuration into the format of the actual PLC configuration. A dialog will open which asks "Convert the configuration to the new format ? Attention: Undo is not possible !" You can select **Yes** or **No**. If you close the dialog with **Yes**, the configuration editor will be closed also. Reopen it and you will see the configuration in the new format. Be aware that after having converted the old format cannot get restored anymore !

6.6.2 General Settings

Select the entry ‘PLC configuration’ (‘root’ module) at top of the configuration tree. Thereupon the dialog ‘Settings’ is shown in the right part of the window:

Calculate addresses: Each newly inserted module automatically is allocated with an address, which results from the address of the module inserted before plus the size of

this address. If a module is removed from the configuration, the addresses of the following modules are adjusted automatically. When the command 'Extras' 'Calculate addresses' is executed, all addresses starting at the selected node (module) are recalculated.

Check for overlapping addresses: At compilation the project will be checked for overlapping addresses and a corresponding message will be displayed.

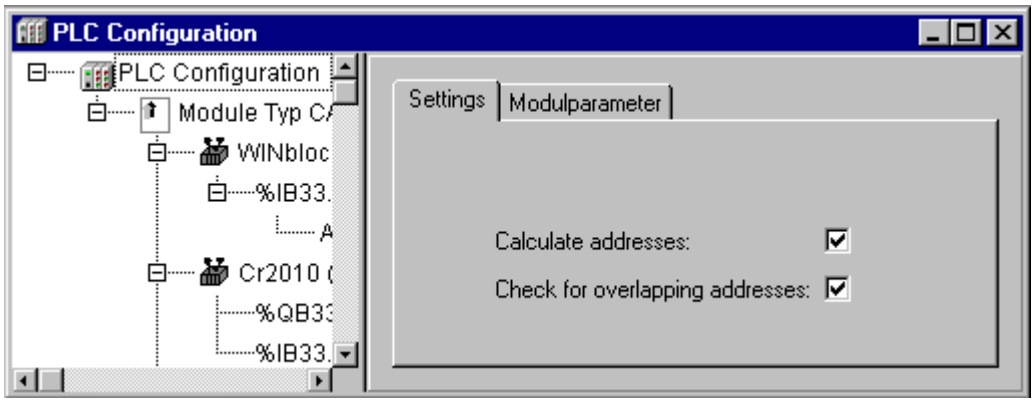


Image 6.8: Dialog for the general Settings of the PLC configuration

The global mode of addressing (flat addresses / addresses depending on Id) in the PLC configuration is defined in the configuration file.

6.6.3 Application-specific parameter dialog (Custom Parameters)

The parametrizing possibilities of the configurator can be expanded by the use of an application-specific DLL, that is an individual dialog. This 'Hook'-DLL must be in that directory which contains the configuration file and then can be linked by an entry in the configuration file.to a module or channel. If done so, for the concerned modules the standard dialog ‘Modulparameter’ will be replaced by a dialog defined in the DLL.

Example:

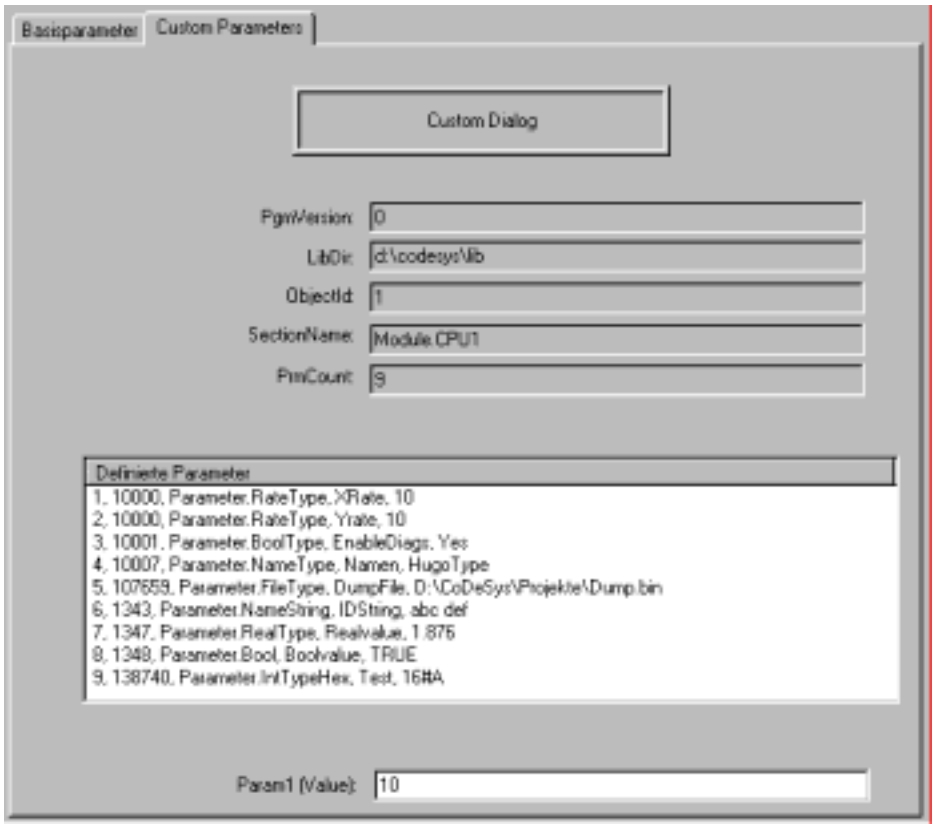


Image 6.9: Example of an application-specific parameter dialog (Custom Parameters)

6.6.4 Configuration of an I/O Module

Basisparameter of an I/O Module

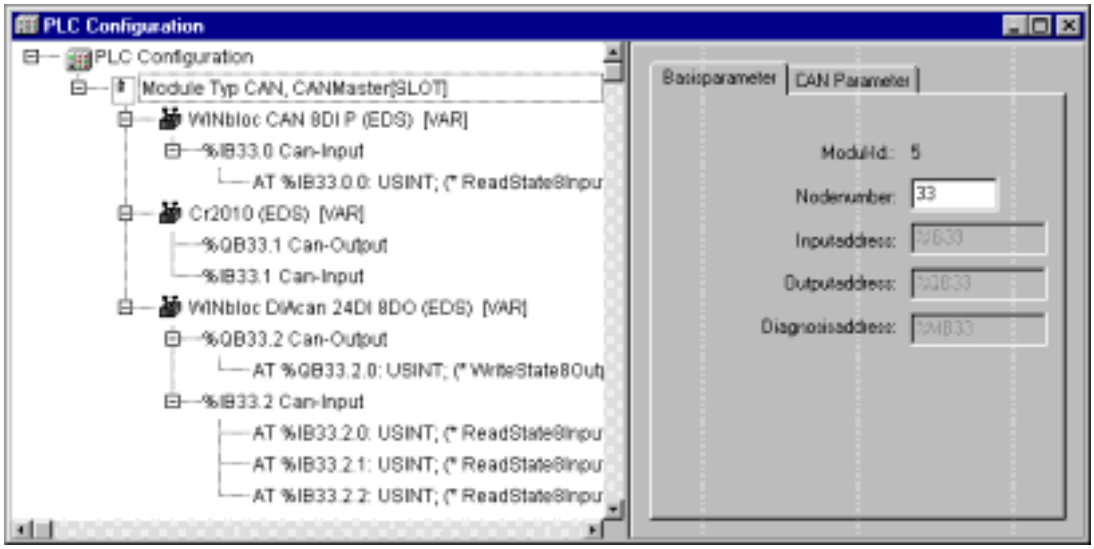


Image 6.11: Basisparameters of an I/O Module

If an I/O module is selected in the configuration tree, the dialog 'Basisparameter' is displayed with the following entries:

Modul-ID: The Modul-ID is a unique identifier of the module within the entire configuration. It is defined by the configuraiton file and it is not editable in the configuration editor.

Nodenumber: The Nodenumber is defined by an entry in the configuration file or – if there is no entry – by the position of the module in the configuration structure.

Inputaddress, Outputaddress, Diagnosisaddress: Addresses for Input- and Output respectively for the storage of diagnosis data.

These addresses refer to the module. It depends on the general settings , which addresses are already predefined, which address mode is valid and whether the addresses can be still edited here.

The diagnosis in the PLC configuration:

A marker address must be given at the **diagnosis address** of the module. For normal I/O modules it depends on the special hardware configuration how the diagnosis will be handled. For bus systems like CAN or Profibus DP the diagnosis works like described in the following: From the given diagnosis address onwards there will be stored various information concerning the structure *GetBusState* which is part of a corresponding library delivered by the manufacturer. All bus modules get a request to fill the diagnosis structure in a cyclic sequence each time when an IEC task has written or read process data to/from the modules. As soon as at least one module in the bus system produces an error, the specific diagnosis information can be read using the function block *DiagGetState* which is also part of the above mentioned library. This function is only available for bus masters, which have been configured within the CoDeSys PLC configuration !

See in the following the input and output parameters of the function block **DiagGetState**. Define an instance of this function block in your **CoDeSys** project to read the diagnosis information for a specific bus module:

Input variables of DiagGetState:

ENABLE: BOOL;	At a rising edge of ENABLE the function block starts working
DRIVERNAME:POINTER TO STRING;	Name of the driver (address of the name) to which the diagnosis request should be sent. If here is entered a 0, the diagnosis request will be forwarded to all present drivers.
DEVICENUMBER:INT;	Identification of the bus which is managed by the driver. E.g.: the Hilscher driver can handle up to 5 cards (busses). The index is 0-based.
BUSMEMBERID:DWORD;	Unique bus-/driver specific identification of the busmodule entifizierung des Busteilnehmers. (E.g.: for a CANopen-card this is the NodeID, for a PB-DP card this is the station address of the participant etc.)

Output variables of DiagGetState

READY:BOOL ;	TRUE: the work on the diagnosis request has been terminated
STATE:INT;	If READY = TRUE then STATE gets one of the

	following values which define the actual state of the function block: -1: invalid input parameter (NDSTATE_INVALID_INPUTPARAM:INT;) 0: function block does not work (NDSTATE_NOTENABLED:INT;) 1: function block is just reading the diagnosis info (NDSTATE_GETDIAG_INFO:INT;) 2: diagnosis info is now available (NDSTATE_DIAGINFO_AVAILABLE:INT;) 3: no diagnosis info is available (NDSTATE_DIAGINFO_NOTAVAILABLE:INT;)
EXTENDEDINFO:ARRAY [0..129] OF BYTE;	Up to 100 Bytes manufacturer specific diagnosis data of the bus. For each bus participant 1 byte is reserved in which the 0 – 2 are used as described in the following: Bit 0: Bus module exists in PLC configuration. Bit 1: Bus module is available in bus system. Bit 2: Bus module reports error.

Modulparameter / Custom Parameters of an I/O Module

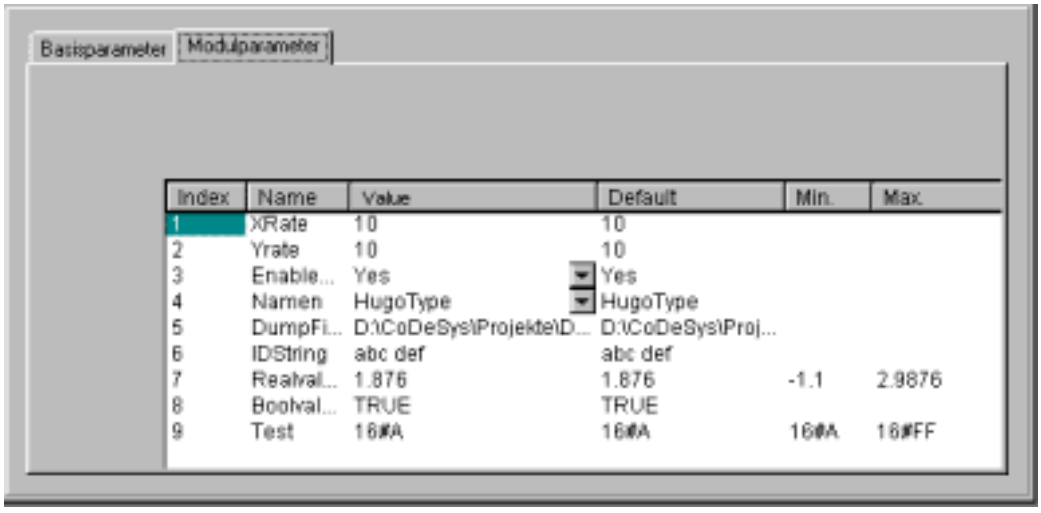


Image 6.10: Modulparameter Dialog

In this dialog the parameters which are given by the device file are shown. Only the column ‘value’ is editable.

Index: The Index is a consecutive digit (i), which numbers through all the way the parameters of the module.

Name: Name of the parameter

Value : Value of the parameter, editable

Initially the default is displayed. Values can be set directly or by means of symbolic names. If the entries in the configuration file are not set to 'Read Only', they can be edited. To do that click on the edit field respectively select on of the entries in a scoll

list. If the value is a file name, you can open the dialog ‘Oben file’ by a doubleclick and browse for another file there.

Default: Default value of the parameters

Min.: minimum value of the parameter (only if no symbolic names are used)

Max.: maximum value of the parameter (only if no symbolic names are used)

A tooltip may give additional information on the currently marked parameter.

Instead of the Modulparameter dialog there might be a customer specific dialog. This is due to the fact, that such a dialog is linked by an entry (Hook-DLL) at the module definition in the configuration file.

6.6.5 Configuration of a channel

Basisparameters of a channel

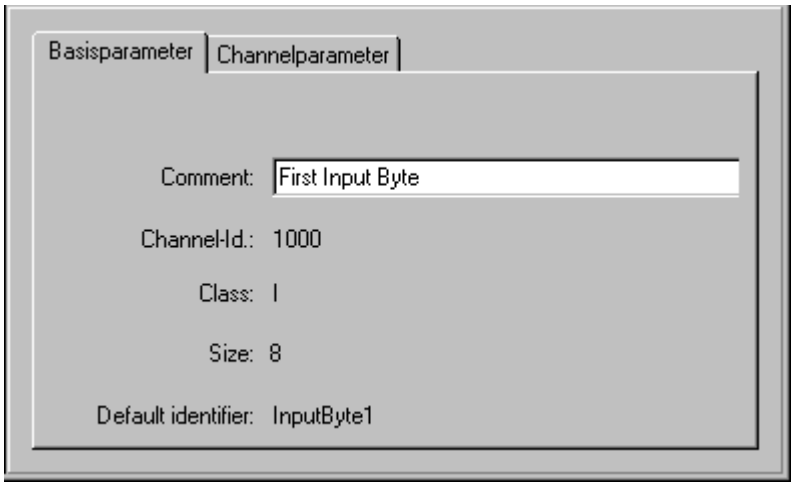


Image 6.11:Basisparameter dialog for a channel

Channel-Id: Globally unique identifier of the channel

Class: Defines whether the channel is used als input (I), output (Q), or as input and output (I&Q), or whether it is switchable (I|Q). If the channel is switchable, this can be done by the command ‘Extras' 'Replace element'.

Size: Size of the channel [Byte]

Default identifier: Symbolic name of the channel

The name of the channel is defined in the configuration file. Only if it is allowed by the definition of the father module, the name of the channel can be edited in the configuration tree.

Comment: Additional information on the channel

In the edit field a comment can be inserted or modified.

Address: This edit field only will be available if it was activated by an entry in the configuration file. Insert the desired address for the channel.

Channelparameter

Corresponding to the Moduleparameter dialog the Channelparameter dialog is used to display and modify the parameters of a channel: **Index**, **Name**, **Wert**, **Default**, **Min**., **Max**. (see Chapter 4.6.4). This dialog also can be replaced by a customer specific dialog 'Custom Parameters' ersetzt sein.

Bitchannels

Bitchannels are automatically inserted, when a channel is defined with an entry CreateBitChannels=TRUE in the configuration file.

The Basisparameter dialog of bitchannels just contains the field **Comment**.

6.6.6 Configuration of Profibus Modules

CoDeSys supports a hardware configuration corresponding to the profibus DP standard. In the profibus system you find master and slave modules. Each slave is provided with a parameter set by ist master and supplies data on request of the master.

A PROFIBUS-DP system consists of one or several masters and their slaves. First the modules must be configured so that a data exchange over the bus is possible. At the initialization of the bus system each master parameterizes the slaves which are assigned to it by the configuration. In a running bus system the master sends and/or requests data to/from the slaves.

The configuration of the master and slave modules in **CoDeSys** is based on the gsd files attached to them by the hardware manufacturer. For this purpose all gsd files which are stored in the configuration files directory will be considered. The modules described by a gsd file can be inserted in the configuration tree and their parameters can be edited there.

Below a master there can be inserted on or several slaves.

If a DP master is selected in the configuration tree, the following dialogs will be available in the right part of the configuration: Basisparameter, DP Parameter, Busparameter, Modulparameter.

If a DP slave is selected, which is inserted below a DP master, the following dialogs will be available: Basisparameter, DP Parameter, Input/Output, Userparameter, Groups, Modulparameter.

If a DP slave is inserted on the level of a master, the following dialogs are available for configuration: Basisparameter, DP Parameter, Input/Output, Modulparameter.

Basic parameters of the DP master

The Basisparameter dialog of a DP master matches that of the other modules: **Module ID, Node number, Input, Output** and **Diagnostic addresses** (see Chapter 4.6.4)

Modulparameters of the DP master

The Modulparameter dialog of a DP master matches that of the other modules: The parameters assigned to the master in additon to the DP and bus parameters in the configuration file are displayed here and the values can be edited in the standard case (see Chapter 4.6.4).

DP parameters of the DP master

This dialog shows the following parameters extracted from the device file of the DP master:

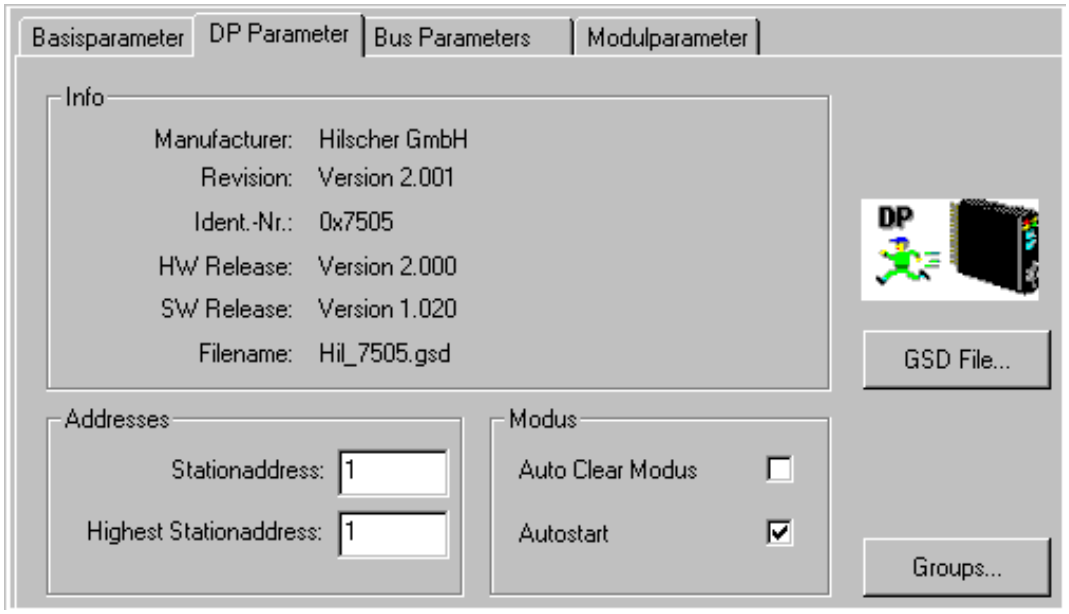


Image 6.12: DP Parameter dialog for DP master

Info	**Manufacturer,** GSD **Revision, ID** (identification number)**, HW Release** and **SW Release** (hard- and software version)**,** GSD-**Filename**
Module name	The settings can be edited at this position.
Addresses	Stationaddress: The allowable range extends from 0 to 126. Each device newly inserted on a bus line is automatically provided the next higher address. (note: Address 126 is the default DP slave address). Manual entry is possible; addresses are tested for duplication. Highest Stationaddress: The highest stationaddress (HSA) assigned on the bus is displayed. Here, a lower address can also be entered in order to narrow the GAP range (that is, the address range within which the search for newly-active devices is carried out).

The GSD file pertaining to a device can be opened and examined using the **GSD File** button.

The **Groups** button leads to the 'Group properties' dialog. The Group properties pertain to the slaves assigned to the master. Up to eight groups can be set up. For each group, enter whether it is to operate in **Freeze mode** and/or **Sync mode**. By assigning slaves (see below: 'Properties of the DP slave' 'Group assignment') to various groups, data exchange from the master can be synchronized via a global control command. With a Freeze command, a master instructs a slave or a group to „freeze“ inputs in their instantaneous state and to transfer these data in the following data exchange. With a Sync command, the slaves are instructed to synchronously switch to output at the next Synch command all data received from the master following the first command.

To switch the Freeze and Sync options for a group on/off, please click with the left mouse button on the appropriate location in the table to place or remove an „X“ next to the desired option, or use the right mouse button to activate or deactivate the option via a context menu. In addition, you can edit the group name here.

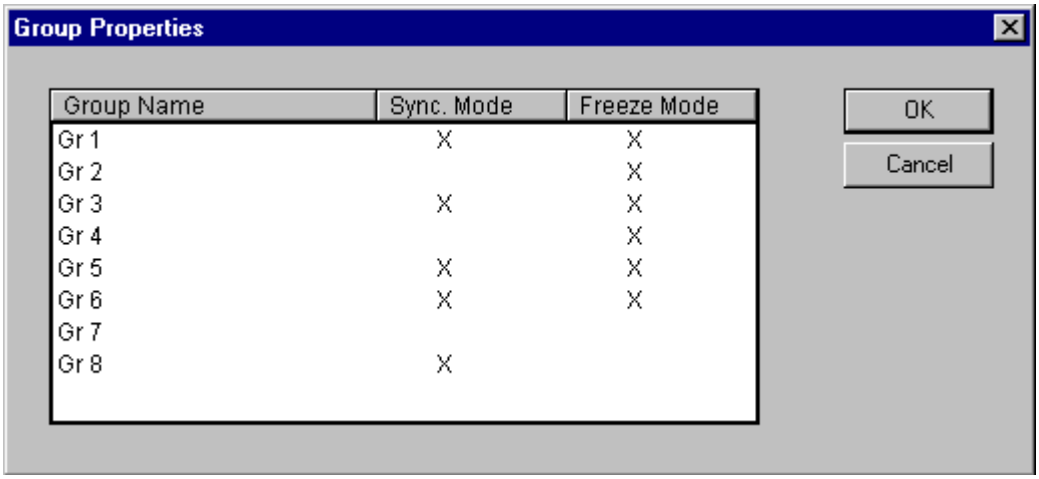


Image 6.13: DP parameters of the DP master / Group Properties

Bus Parameters of the DP master

The Bus parameters describe the timing of the communication. If the option **Optimize** is activated, then the parameter values will be calculated automatically depending on the **Baudrate** set by the user and the settings given by the GSD files.

Attention: The automatically calculated values are just approximated values !

All parameters can also be edited manually.

Baud rate	The entries already present in the GSD file are available for selection, but only a transmission rate supported by all slaves can be entered.
Optimize	If the option is activated, the entries made in the 'Bus parameters' dialog will be optimized with respect to the specifications in the GSD files; it is only possible to edit the

	values if the option is deactivated. **Important:** The values calculated automatically are only rough approximate values.
Slot Time	Maximum time during which the master waits, after sending a request message, for the receipt of the first character of the slave's reply message
Min.Station Delay	min. TSDR (in tbit): minimum reaction time after which a station on the bus may reply (min. 11 tBit)
Max.Station Delay	max. TSDR (in tbit): maximum time span within which a slave must reply.
Quiet Time	TQUI (in tbit): idle period which must be taken into account during conversion of NRZ (Non Return to Zero) signals to other codings (switchover time for repeater)
Target Rotation Time	TTR (in tbit): token cycle time setting; projected time interval in which a master should receive the token. Result of the sum of the token stop times of all masters on the bus.
Gap Update Factor	GAP update factor G: number of bus cycles after which the master's GAP (address range from its own bus address to the address of the next active station) is searched for an additional, newly inserted active station.
Max. Retry Limit	maximum number of repeated request attempts by the master when it has not received a valid response from the slave
Min. Slave Interval	Time between two bus cycles in which the slave can process a request from the master (time basis 100µs). The value entered here must be checked against the respective specifications in the slave's GSD file.
Poll Timeout	Maximum time after which the master's reply by a master-master communication must be retrieved by the requester (Class 2 DP master) (time basis 1 ms).

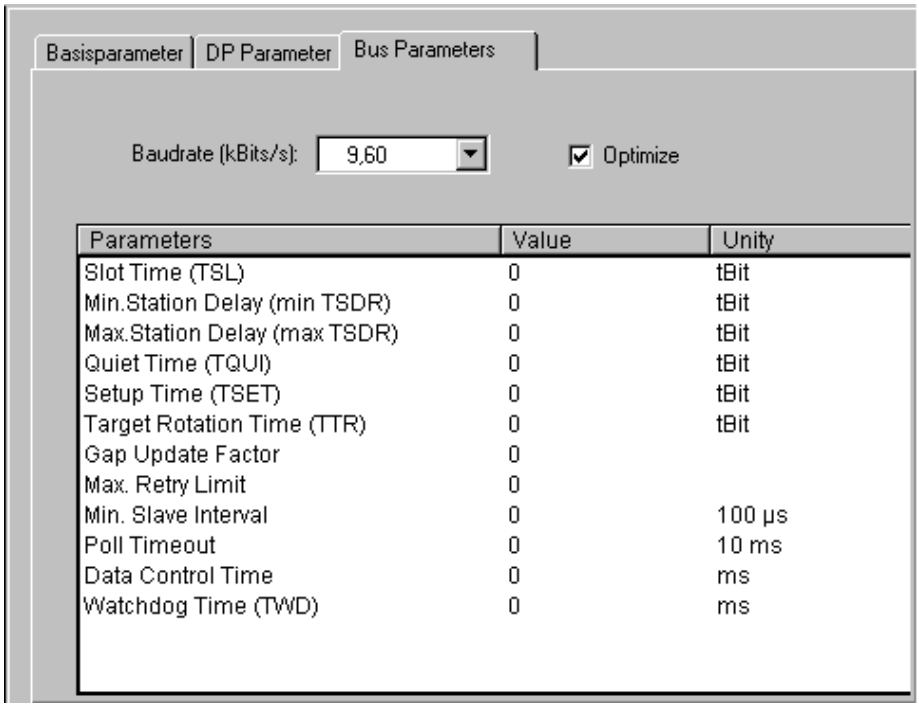


Image 6.14: Bus Parameter of the DP-Master

Data Control Time Time in which the master reports its status to the slaves assigned to it. At the same time, the master monitors whether at least one data exchange each has taken place with the slaves within this period, and updates the Data_Transfer_List.

Watchdog Time Time value for the access monitoring (watchdog). Setting is currently not supported (fixed-set to 400 ms)

Basic parameters of a DP slave

The Basisparameter dialog of a DP-Slaves corresponds to that of other modules: **Modul-Id**, **Nodenumber**, **Input-**, **Output-** and **Diagnosisaddress** (see Chapter 4.6.4).

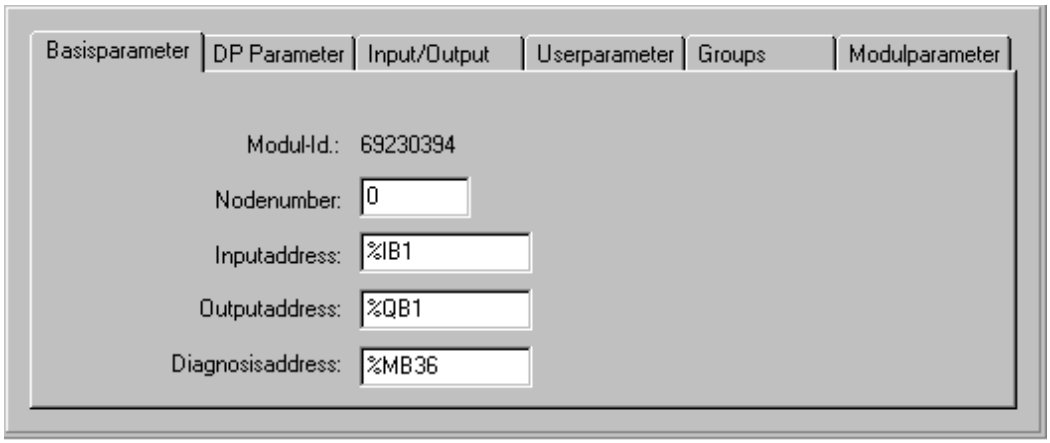


Image 6.15: Basisparameter dialog for a DP slave

DP Parameters of a DP slave

This dialog shows the following parameters extracted from the device file of the DP slave (You can inspect the corresponding GSD file via the **GSD-File** button):

Info	**Manufacturer**, GSD **Revision**, **HW Release** and **SW Release** (hard- and software version), GSD-**Filename**, Slavetype
Standardparameter	**Identnumber**: Unique identnumber assigned by the PNO for this device type. Allows unambiguous reference between a DP slave and the corresponding GSD file. **TSDR (Tbit*)**: Time Station Delay Responder: Reaction time, the earliest time after which the slave is permitted to respond to the master. (min. 11 TBit) * TBit: Time unit for transfer of a bit on the PROFIBUS; Reciprocal value of the transimission rate; z.B. 1 TBit at 12MBaud=1/12.000.000 Bit/sek=83ns **Lock/Unlock**: Slave is locked or released to other masters: 0: min.TSDR and slave-specific parameters may be overwritten 1: Slave released to other masters, 2: Slave locked to other masters, all parameters are accepted; 3: Slave released to other masters
Identification	**Station address** (see 'Properties of the DP masters'), **Station name** (matches device name, editable)
Activation	Slave is active/inactive in current configuration. If activation is not selected, configuration data will still be transferred to the coupler on Download, but not data exchange occurs over the bus.
Watchdog	If **Watchdog Control** is set active, the entered Watchdog time applies (access monitoring, basis 10 ms). If the slave has not been accessed by the master within this time, it is reset to its initialization state.

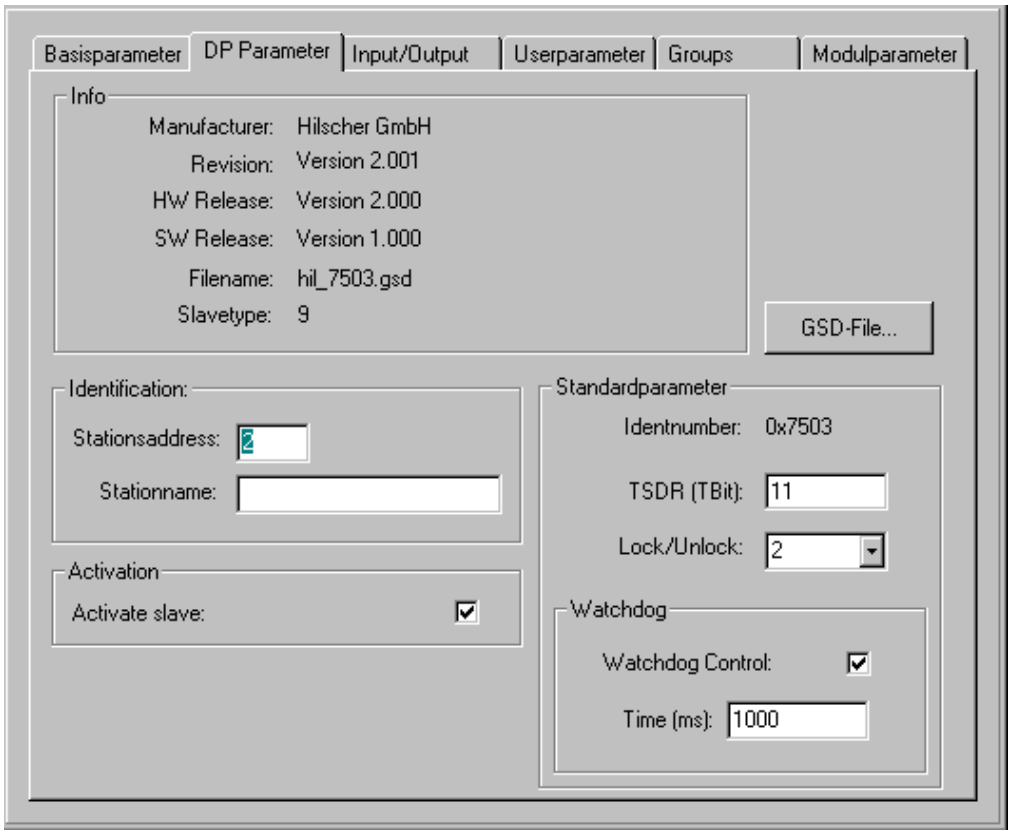


Image 6.16: DP Parameter dialog for a DP slave

Inputs/Outputs of a DP slave

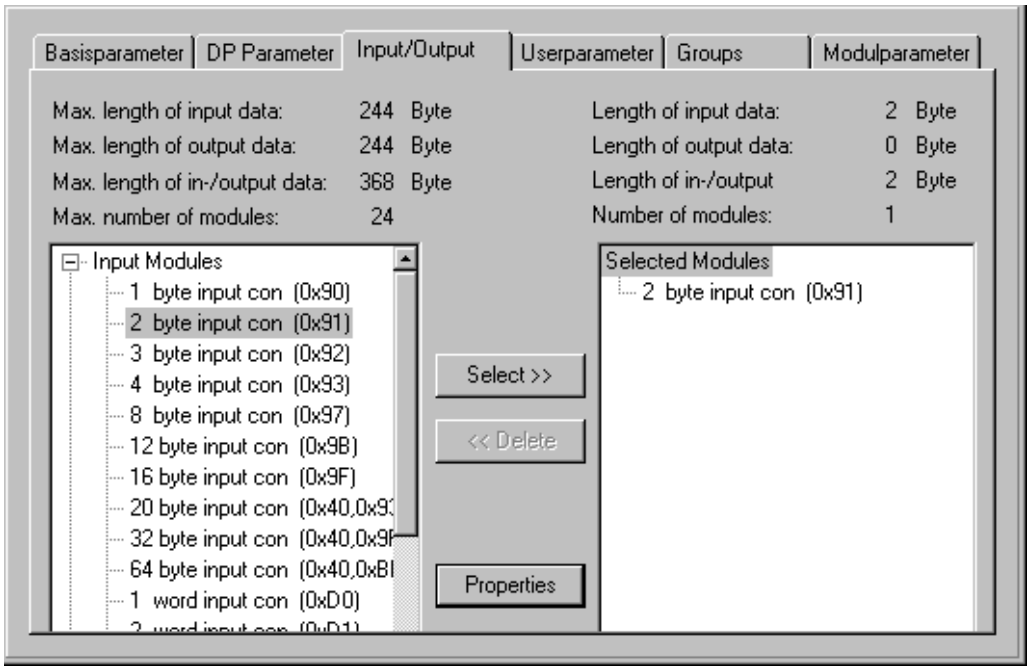


Image 6.17: Dialog for configuring the in-/outputs of a DP slave

As the maximum data lengths specified in the GSD file (**Max. length of input data, Max. length of output data, Max. length of in-/output data**) and the maximum number of modules (**Max. number of modules**) must be respected, this information is displayed in both module lists. The left block displays the maximum possible

values for the device, the right the values resulting from summing the values selected in the configuration. If the maximum values are exceeded, an error message is issued.

The dialog lists in the left window all the in- and output modules available in the slave's GSD file, while the right window contains the configuration currently selected for this device as it relates to in- and outputs.

If this is a modular slave (a device that can be equipped with various I/O modules), the selection proceeds as follows: In the left-hand list, the desired in- or output module is selected by mouse-click and copied into the right window using the Select >> button. Incorrect entries can be corrected by selecting the undesired module in the right window and pressing the **Delete** button.

This kind of selection is not possible for non-modular slaves. These directly enforce a closed display of their in- and outputs in the right window. Undesired modules can then by removed by selecting and using **Delete**.

The **Properties** button leads to the 'Module properties' dialog for the in- or output module currently selected in the left or the right list. It shows the **Name**, the **Config** (module description coding according to PROFIBUS standard) and the **in-** and **output lengths** of the module in **bytes**. If the module description in the GSD file contains specific parameters in addition to the standard set, these are listed here with their values and range of values. If the **Symbolic names** option is activated, the symbolic names are then used.

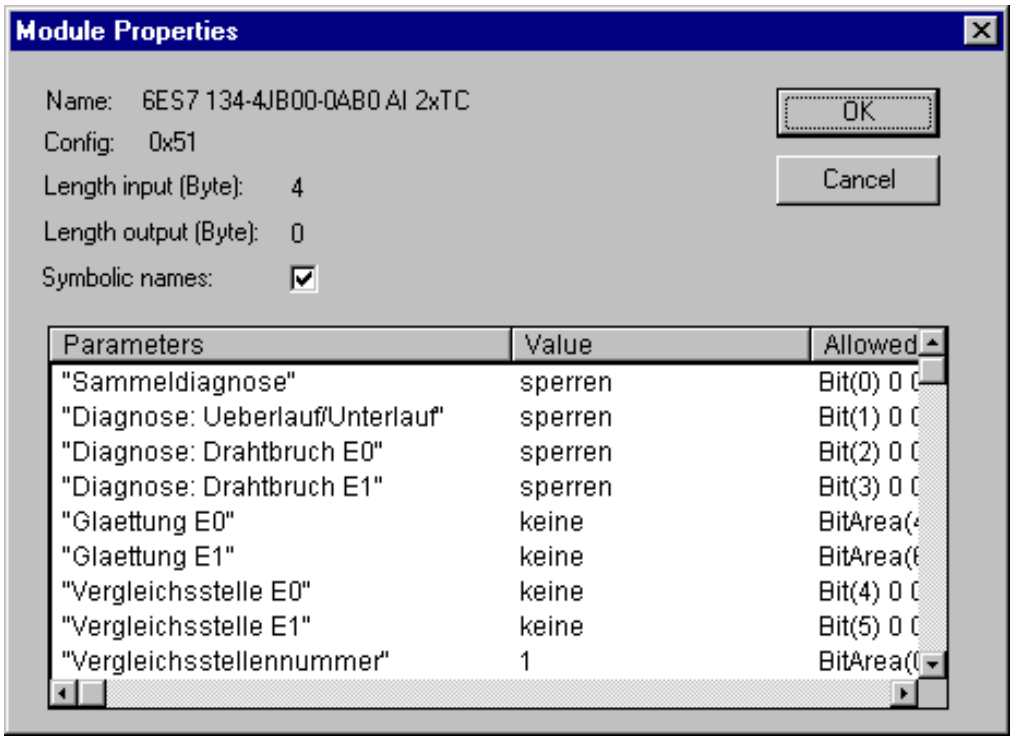


Image 6.18: Module properties dialog for in-/outputs of a DP slave

User parameters of a DP slave

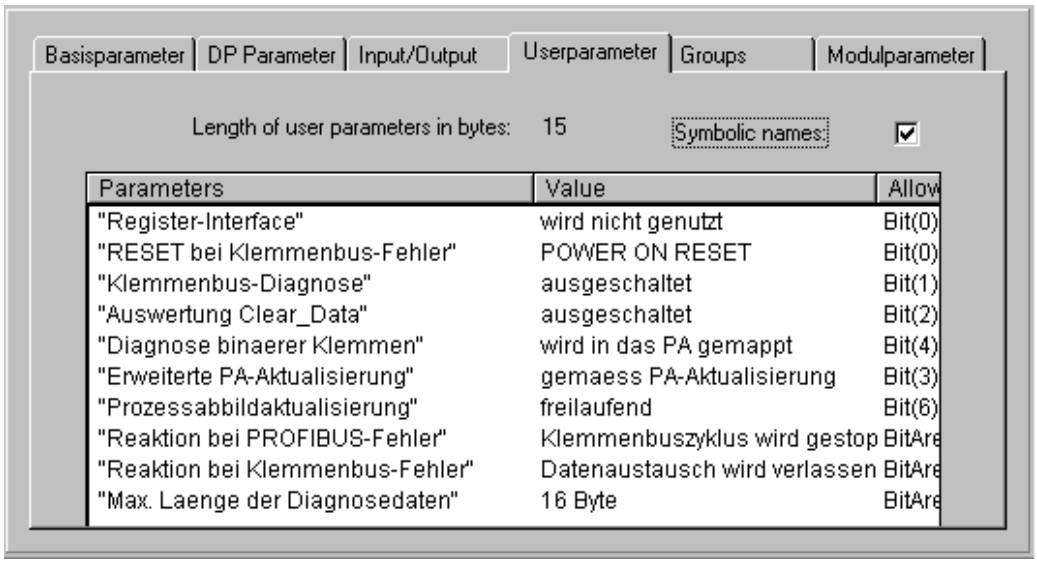


Image 6.19: Userparameter dialog for a DP slave

Here, various extended parameters of a DP slave, defined in the GSD file, are listed. The **Parameters** column shows the name of the parameter. The parameter values entered in **Value** column can be altered by double-click or via the right mouse button. In addition, the **Value range** is specified.

If symbolic names are also specified for the parameters in the GSD file, the **Symbolic names** option can be activated, so that the values can be displayed with these names. For information, the **Length of user parameters** is also given above the table.

Group assignment of a DP slave

This dialog is used for assigning the slave to one or more of the eight possible groups. The universally applicable group properties (**Sync. Mode** and/or **Freeze Mode**), on the other hand, are defined during configuration of the master's properties (see 'Properties of the DP master, Group properties'). This dialog can also be reached via the **Global Group Properties** button.

The group(s) to which the slave is assigned are marked with a plus sign. The assignment to or removal from a group is accomplished by selecting the group name in the **Group Membership** column and pressing 'Add slave to group' or 'Remove slave from group' with the right mouse button, or by clicking again with the mouse to the left of the group name.

A slave device can only be assigned to those groups whose properties it supports. The concerned properties of each slave (**Sync. Mode** / **Freeze Mode**) are displayed above the table. The modes supported by the device are checked.

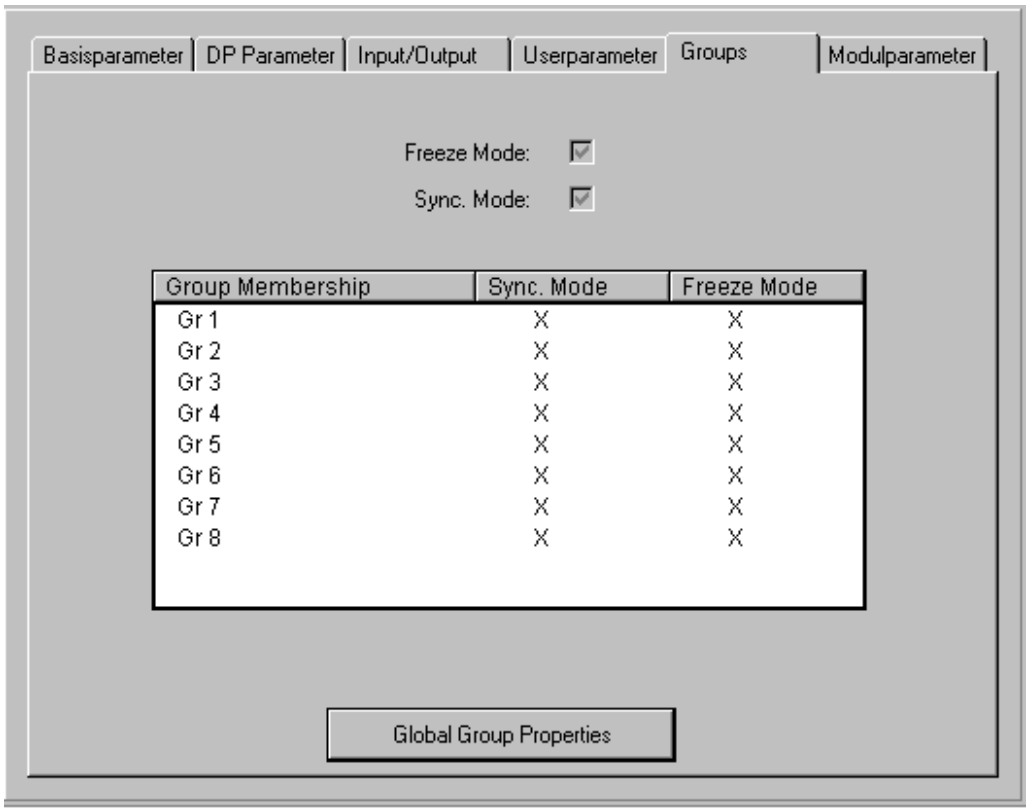


Image 6.20: Dialog for group assignment of a DP slave

Modulparameter

The module parameters dialog of a DP slave matches that of the other modules: The parameters assigned to the slave in addition to the DP and user parameters in the configuration file are displayed here, and the values can be edited in the standard case.

Properties of a DP slave in slave operation of the Profibus

If a Profibus runs in slave mode, the slave device is inserted in the master level of the configuration tree. The configuration can be done in the following dialogs:

- Basisparameter (see above, Basisparameter of a DP slave)
- DP Parameter (see above, DP Parameter of a DP slave)
- Modulparameter (see above, Modulparameter of a DP slave)
- Input/Output

6.6.7 Configuration of CAN modules

CoDeSys supports a hardware configuration according to CANopen Draft Standard 301. The configuration looks like that described for the hardware dependant configuration.

All EDS (Electronic Data Sheet) respectively DCF (Device Configuration File) files which are stored in the subdirectory PLCCONF of the library when CoDeSys is

started, can be integrated, edited and displayed in the configuration. In the EDS file the configuration options of a CAN module are described. If you add a module, which is described in a DCF file, only the IEC addresses can be modified, for the module has already been configured completely in a CAN configurator.

Basis Parameters of a CAN Master

For information on **Modul-Id, Input-/Output- and diagnosis address** see Chapter 4.6.4.

CAN Parameters of a CAN Master

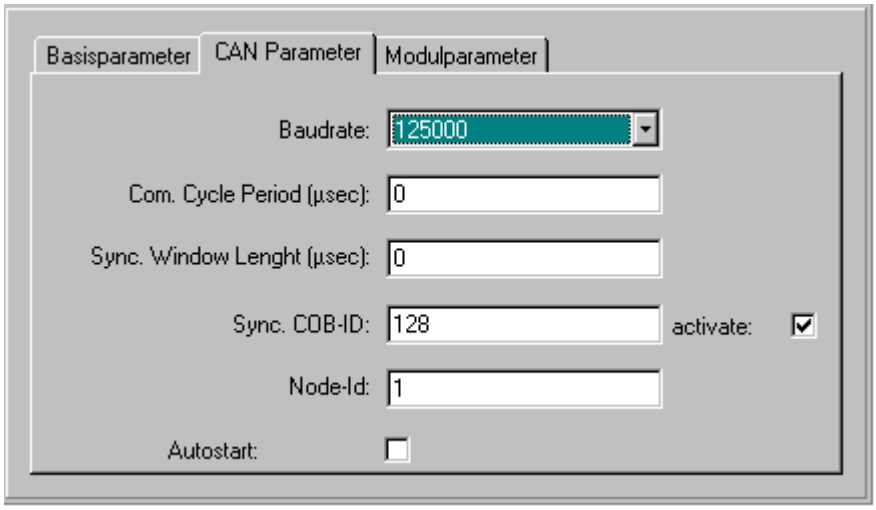
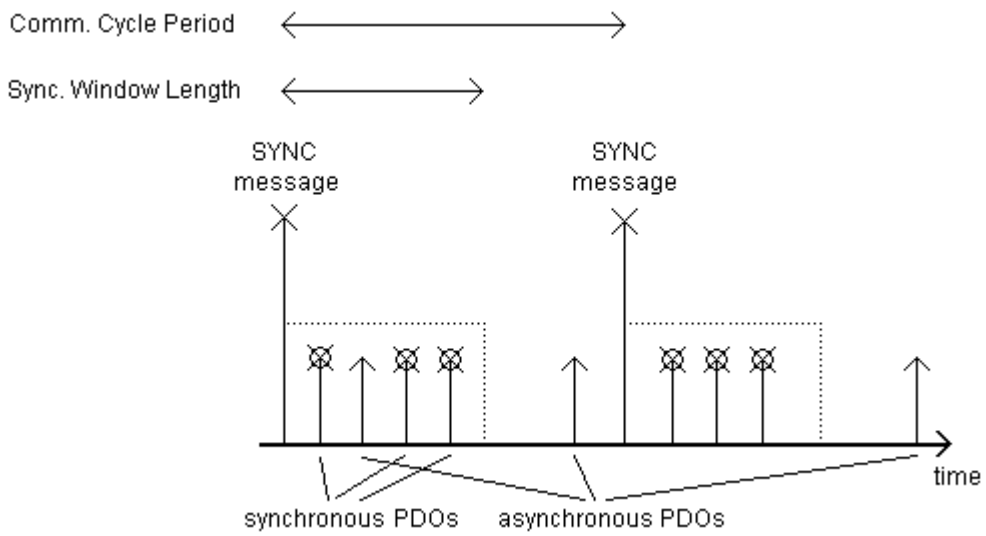


Image 6.21: CAN Parameter Dialog for a CAN Master

The properties for transmission on the CAN bus can be set directly after the insertion of the module or can be called up with the command **'Extras' 'Properties'**.

Using the selection option, set the required **Baud rate** which the transmission should take place at.

One differentiates between synchronous and asynchronous transmission modes (see PDO properties) for PDO's (Process Data Object). The synchronisation message is sent with a unique number **Sync.COB-Id** (Communication Object Identifier) in the interval in microseconds which is given by the **Communication Cycle Period**. The synchronous PDO's are transmitted directly after the synchronisation message in the time window (**Sync.Window Length** in microseconds). No synchronisation message will be sent if the fields Comm. Cycle Period and Sync. Window Length contain 0.



activate: Only if this option is activated, an intermodule exchange of synchronization messages will be possible.

NodeID: serves to identify the CAN module uniquely and corresponds to the set number on the module itself which is between 1 and 127. The Id must be entered as a decimal number. (Do not mix up with the 'Node number' !)

The CAN bus will automatically initialised and started when downloading is occurring and when the controller system starts up if the option **Automatic start** is activated. The CAN bus must be started up in the project if this option is not active.

If the option **Support DSP301,V3.01 and DSP306** is activated, then modular CAN Slaves as well as some additional extensions concerning the standards DSP301 V3.01 and DSP306 will be supported. In this case e.g the stroke of the Heartbeat will be adjustable (**Heartbeat Master [ms]:**). Working with Heartbeats is an alternative guarding mechanism: In contrast to the Nodeguarding functionality it can be executed by Master- **a**nd Slave-Modules. Usually the master will be configured to send heartbeats to the slaves.

Module parameter of a CAN master

The modul parameters dialog of a CAN corresponds to that for the other modules: The parameters which have been additionally assigned to the master in the configuration file, are displayed here and as a default the values can be edited. (see Chapter 4.6.4)

Basis parameters of a CAN module

For information on **Modul-Id, Input-/Outputadresses** see Chapter 4.6.4.

CAN Parameters of a CAN Module

The CAN parameters of a CAN module, which is not acting as master (global watching of the bus), are different to those of a CAN master.

Section General:

The **Node-Id** serves to identify the CAN module uniquely and corresponds to the set number on the module itself which is between 1 and 127. The Id must be entered as a decimal number.

If **DCF write** is activatedm a DCF file will be created after inserting an EDS file in the defined directory for the compiled files whose name is made up of the name of the EDS file and the Node Id which is tacked on the end.

If the option **Create all SDO's** is activated, then for all objects SDO's will be created (not only for those that have been modified).

If the option **Reset node** is activated, then the slave will be reset before downloading the configuration.

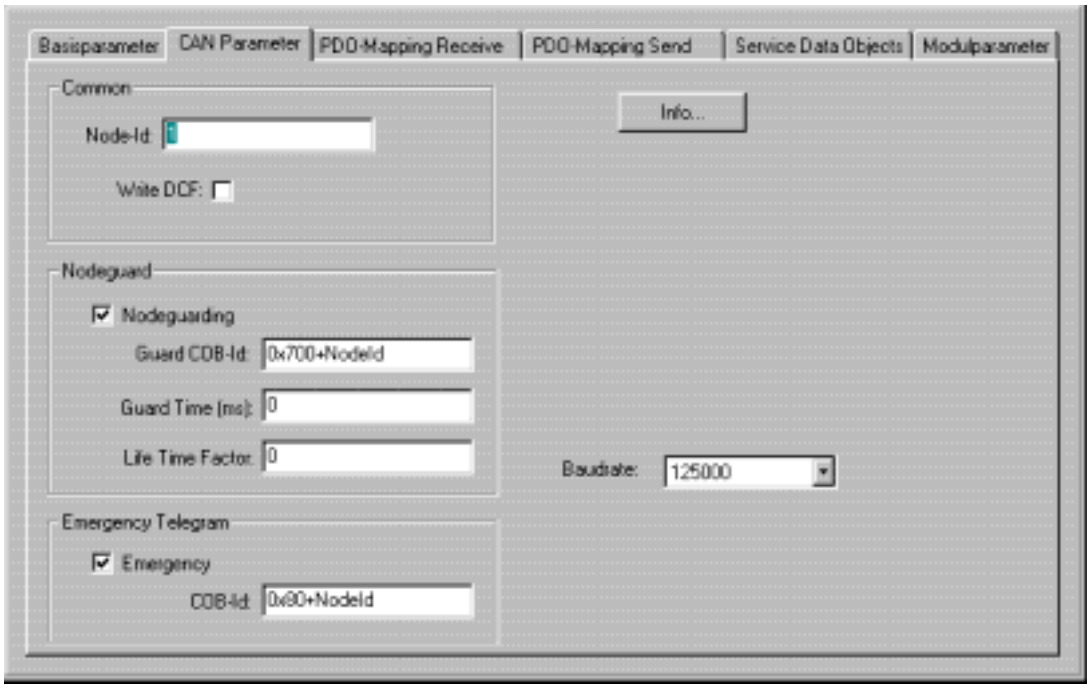


Image 6.22: CAN Parameter Dialog für einen CAN-Moduls

Section Nodeguard Settings: (alternatively to guarding by the Heartbeat mechanism)

If the option **Nodeguarding** is activated, a message will be sent to the module in the interval which is set by **Guard Time** in milliseconds. If the module does not then send a message with the given **Guard COB-ID** (Communication Object Identifier), it receives the status timeout. If the number of attempts (**Life Time Factor)** is reached the module receives the status not OK. The status of the module will be stored at the diagnosis address (see above). No monitoring of the module occurs if the variables Guard Time and Life Time Factor are not defined (0).

Section Heartbeat Settings: (alternatively to Nodeguarding)

If the option **Activate Heartbeat Producer** is activated, the module will send heartbeats according to the interval defined in **Heartbeat Producer Time**: given in **ms**.

If the option **Activate Heartbeat Consumer** is activated, then the module will listen to heartbeats which are sent by the master. As soon as no more heartbeats are received, the module will switch off the I/Os.

Section Emergency Telegram:

A module sends an **emergency** message, with a unique **COB-Id**., when there is an internal error. These messages, which vary from module to module, are stored in the diagnosis address.

The entries “FileInfo” and “DeviceInfo” of the EDS or DCF file from the corresponding module manufacturer are hidden behind the **Info** button.

CAN Modules Selection for Modular Slaves

In the left column (**Available modules**) you find all modules which are available for the slave. Mark the desired modules and by using the buttons **Add** and **Remove** create a selection in the right column (**Selected Modules**). The PDO- and SDO selection will be updated automatically.

PDO mapping of a CAN module

The tabs **Receive PDO mapping** and **Send PDO mapping** in the configuration dialog for a CAN module allow the ”mapping” of the module, which is described in the EDS file, to be changed.

All of the ”mappable” objects in the EDS file are located on the left side and can be added in the right side to the PDO’s (Process Data Object) (”**>>**” button) or removed again (**Remove** button). The StandardDataTypes can be inserted to create empty spaces in the PDO.

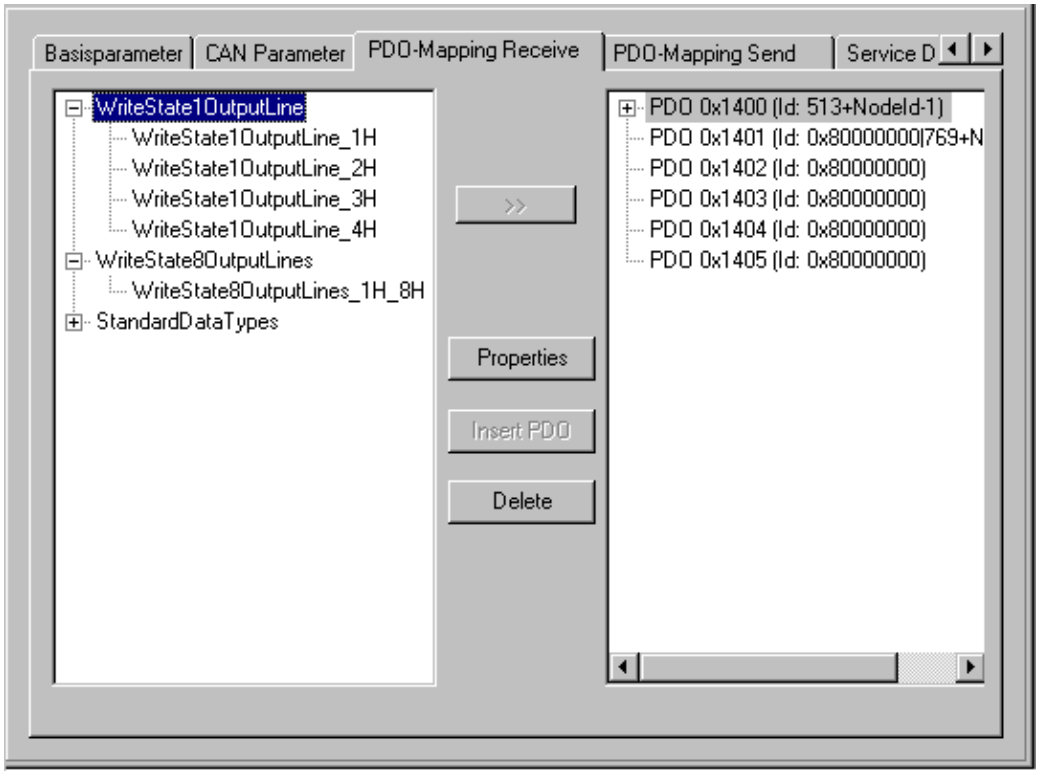


Image 6.23: Configuration of a CAN module, PDO-Mapping send

The button **Insert Element** can be used to create further PDO’s and to add appropriate objects to them. The allocation of inputs or outputs to the IEC addresses can be achieved over the inserted PDO’s. The setting which have been made in the

controller system configuration will become visible when one leaves the dialog. The individual objects can be afforded symbolic names there.

The standard set properties of the PDO's can be edited using **Properties**.

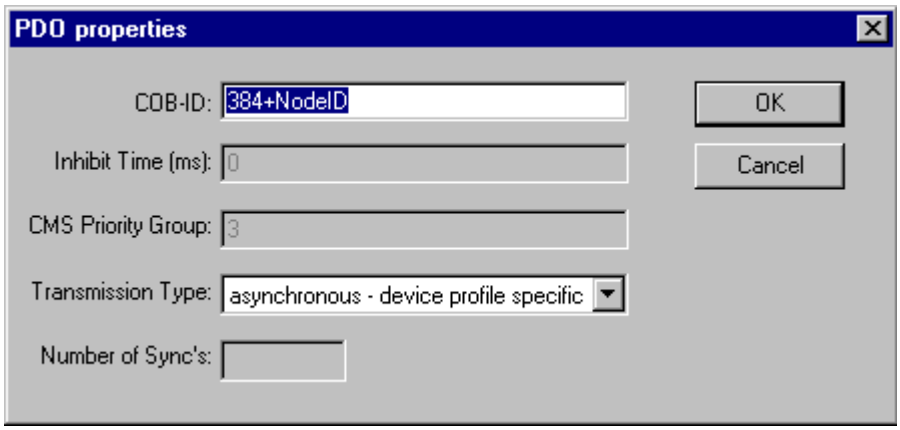


Image 6.24: PDO Properties dialog

Each PDO message requires a unique **COB-Id** (Communication Object Identifier).

The field appears in grey and cannot be edited if an option is not be supported by the module or if the value cannot be changed.

The **Inhibit Time** is the minimum time between two messages from this PDO. This is to prevent PDO's which are sent when the value is changed from being sent too often.

The **CMS Priority Group** cannot be changed and describes the relative importance of the PDOs during the CAN transmission. Values from 0 to 7 are displayed, whereby 0 is the highest.

Transmission Type offers you a selection of possible transmission modes for this module:

- **acyclic - synchronous**: the PDO will be transmitted synchronously but not periodically.
- **cyclic – synchronous:** the PDO will be transmitted synchronously, whereby the **Number of Sync's** gives the number of synchronisation messages, which lie between two transmissions of this PDO.
- **synchronous – RTR only**: the PDO will be updated after each synchronisation message but not sent. It is only sent when there is an explicit request to do so (**R**emote **T**ransmission **R**equest)
- **asynchronous – RTR only**: the PDO will only be updated and transmitted when there is an explicit request to do so (**R**emote **T**ransmission **R**equest)
- **asynchronous – device profile specific and asynchronous - manufacturer specific**: the PDO will only be transmitted when specific events occur.

Number of Syncs: If cyclic transmission has been set, enter here the number of synchronisation messages (see 'Com. Cycle period' in the CAN parameter dialog which should be sent between two transmissions of the PDO.

Event-Time: If an corresponding transmission type is set, enter here in milliseconds (ms) the interval between two transmissions.

Service Data Objects

Here you find a list of all objects in the EDS or DCF file which are in the area of the Index 0x2000 to 0x9FFF and which are marked as writable.

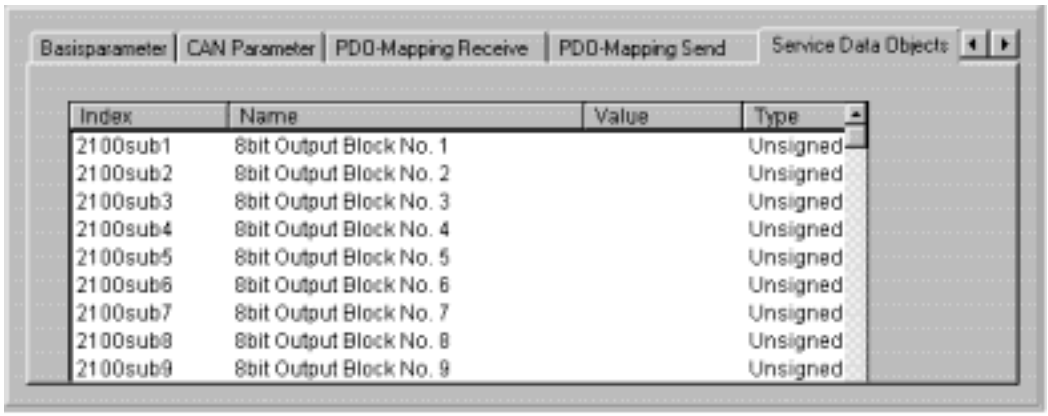

Basisparameter
CAN Parameter
PDO-Mapping Receive
PDO-Mapping Send
Service Data Objects
Index
Name
Value
Type
2100sub1 8bit Output Block No. 1 Unsigned
2100sub2 8bit Output Block No. 2 Unsigned
2100sub3 8bit Output Block No. 3 Unsigned
2100sub4 8bit Output Block No. 4 Unsigned
2100sub5 8bit Output Block No. 5 Unsigned
2100sub6 8bit Output Block No. 6 Unsigned
2100sub7 8bit Output Block No. 7 Unsigned
2100sub8 8bit Output Block No. 8 Unsigned
2100sub9 8bit Output Block No. 9 Unsigned

Image 6.25: Dialog for configuration of the Service Data Objects (SDO)

The properties **Index, Name, Value, Type** and **Default** are displayed for every object. The value can be changed. Mark the value and press the <Space bar>. After making the change confirm the new value with <Enter> or reject it with the <Escape> key.

The set values are transmitted in the form of SDO's (Service Data Object) to the CAN modules at the initialisation of the CAN bus.

Note: All incompatible data types between CANopen and IEC-61131 will be replaced in CoDeSys by the next larger IEC-61131 data type.

PLC Configuration in online mode

In online mode the PLC configuration displays the stati of the inputs and outputs of the PLC. If a boolean input or output has the value TRUE, the little box at the beginning of the entry line in the configuration tree will get blue, non-boolean values will be added at the end of the entry line (e.g. "=12").

The boolean inputs can be toggled by mouseclicks. At other inputs a dialog opens, where the value can be modified. The modified value will be set in the PLC as soon as the dialog is closed with **OK**.

6.7 Task Configuration

In addition to declaring the special PLC_PRG program, you can also control the processing of your project using the task management.

A **Task** is a time unit in the processing of an IEC program. It is defined by a name, a priority and by a type determining which condition will trigger the start of the task. This condition can be defined by a time (cyclic, freewheeling) or by an internal or external event which will trigger the task; e.g. the rising edge of a global project variable or an interrupt event of the controller.

EINBETTENFor each task you can now specify a series of programs that will be started by the task.

The combination of priority and condition will determine in which choronological order the tasks will be executed.

For each task you can configure a watchdog (time control) can be configured.

In the Online Mode the task processing can be monitored in a diagram.

Additionally there is the possibility to link **System events** (e.g. Start, Stop, Reset) directly with the execution of a project POU.

The Task Configuration is found as an object in the **Resources** tab the Object Organizer. The **Task editor** is opened in a bipartited window.

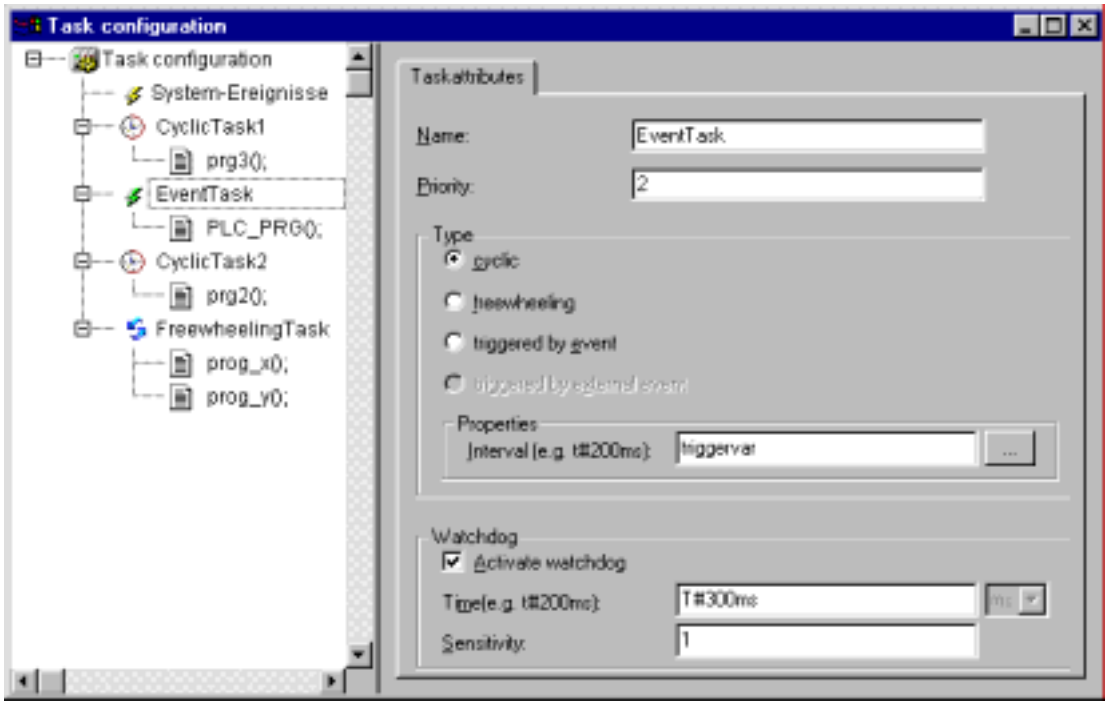


Image 6.26: Example for a Task Configuration

In the left part of the window the tasks are represented in a **configuration tree**. At the topmost position you will always find the entry 'Taskconfiguration'. Below there are the entry 'System events' and the entries for the particular tasks, represented by the task name. Below each task entry the assigned program calls are inserted. Each line is preceeded by an icon.

In the right part of the window a **dialog** will be displayed which belongs to the currently marked entry in the configuration tree. Here you can configure the tasks (Task properties), program calls (Program call) resp. define the linking of system events (System events). It depends on the target which options are available in the configuration dialogs. They are defined by a description file which is referenced in

the target file. If the standard descriptions are extended by customer specific definitions, then those will be displayed in an additional tab 'Parameter' in the right part of the window.

Note: Please do not use the same string function (see standard.lib) in several tasks, because this may cause program faults by overwriting.

Working in the Task Configuration

- The most important commands you find in the context menu (right mouse button).
- At the heading of the Task Configuration are the words "Task Configuration." If a plus sign is located before the words, then the sequence list is closed. By doubleclicking on the list or pressing <Enter>, you can open the list. A minus sign now appears. By doubleclicking once more, you can close the list again.
- With the **'Insert' 'Insert Task'** command, you can insert a task.
- With the **'Insert' 'Append Task'** command, you can insert a task at the end of the configuration tree.
- With the **'Insert' 'Insert Program Call'**, a program call will be assigned to the task which is actually selected in the configuration tree.
- Furtheron for each entry in the configuration tree an appropriate configuration dialog will appear in the right part of the window. There options can be activated/deactivated resp. inputs to editor fields can be made. Depending on which entry is selected in the configuration tree, there will be the dialog for defining the 'Taskattributes' (see 'Insert Task'), the dialog for defining a 'Program Call' (see 'Insert Program Call') or the table of 'System event'. The settings made in the dialogs will be taken over to the configuration tree as soon as the focus is set to the tree again.
- A task name or program name can also get edited in the configuration tree. For this perform a mouseclick on the name or select the entry and press the <Space> button to open an edit frame.
- You can use the arrow keys to select the previous or next entry in the configuration tree.
- By clicking on the task or program name, or by pressing the <Space bar>, you can set an edit control box around the name. Then you can change the designation directly in the task editor.

‘Insert‘ ‘Insert Task‘ or ‘Insert‘
‘Append Task‘

With this command you can insert a new task into the Task Configuration. The entries each consist of a symbol and the task name.

If a task or the entry 'System events' is selected, then the **'Insert Task'** command will be at your disposal. The new task will be inserted in front of the cursor. If the words Task Configuration are selected, then the **'Append Task'** is available, and the new task will be appended to the end of the existing list.

The dialog box will open for you to set the task attributes.

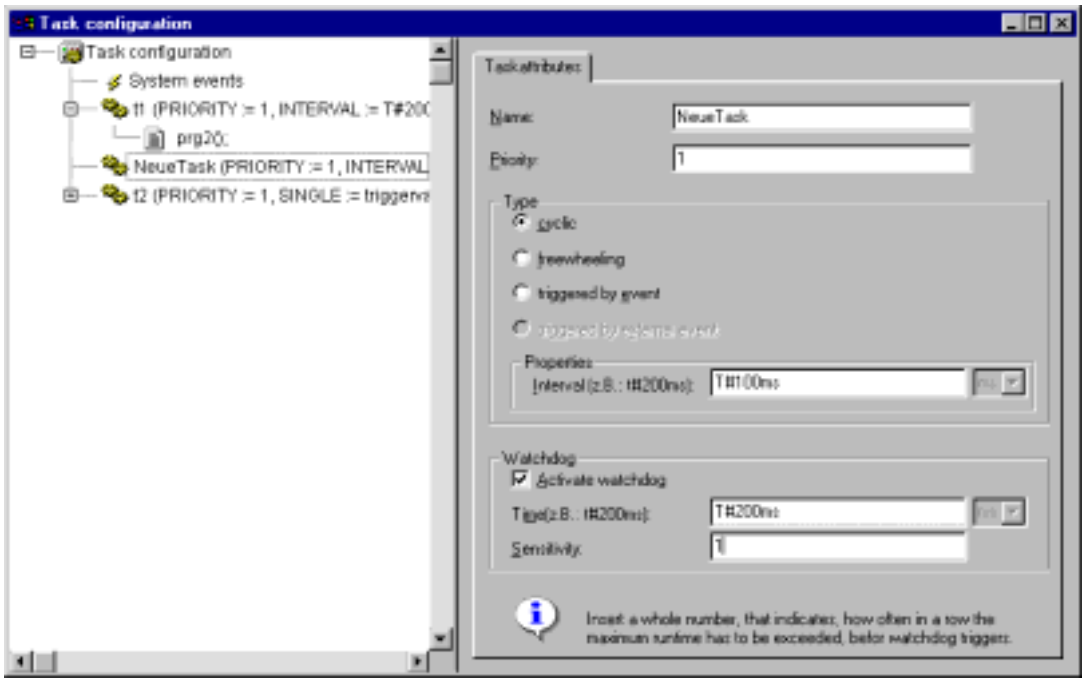


Image 6.27: Dialog Box for Setting Task Attributes

Insert the desired attributes:

Name: a name for the task; with this name the task is represented in the configuration tree; the name can be edited there after a mouseclick on the entry or after pressing the <Space> key when the entry is selected.

Priority (0-31): (a number between 0 and 31; 0 is the highest priority, 31 is the lowest),

Type:

cyclic ( ) : The task will be processed cyclic according to the time definition given in the field 'Interval' (see below).

freewheeling ( ) : The task will be processed as soon as the program is started and at the end of one run will be automatically restarted in a continuous loop. There is no cycle time defined.

triggered by event ( ): The task will be started as soon as the variable, which is defined in the Event field gets a rising edge.

triggered by external event ( ): The task will be started as soon as the system event, which is defined in the Event field, occurs. It depends on the target, which events will be supported and offered in the selection list.

Properties:

Interval (for Type 'cyclic' or 'freewheeling'): the period of time, after which the task should be restarted. If you enter a number, then you can choose the desired unit in the selection box behind the edit field: milliseconds [ms] or microseconds [µs]. Inputs in [ms]-format will be shown in the TIME format (e.g. "t#200ms") as soon as the window gets repainted; but you also can directly enter the value in TIME format. Inputs in [ms] will always be displayed as a pure number (e.g. "300").

Single (for Type 'triggered by event' or 'triggered by external event'): a global variable which will trigger the start of the task as soon as a rising edge is detected. Use button ... or the input assistant <F2> to get a list of all available global variables.

If there is no entry in both of these fields, then the task interval will depend on which runtime system is used (see runtime documentation); e.g. in this case for CoDeSys SP NT V2.2 and higher an interval of 10 ms will be used).

Watchdog

Activate watchdog When this option is activated (☑) then the task will be terminated in error status as soon as the processing takes longer than defined in the 'Time' field (see below).

Time (e.g.: t#200ms): Watchdog time; after the expiration of this term the watchdog will be activated unless the task has not been terminated already.

Sensitivity: Number of overruns of the watchdog time, which are accepted without generating an error.

‘Insert‘ ‘Insert Program Call‘ or ‘Insert‘‘Append Program Call‘

With these commands you will open the dialog box for entering a program call to a task in the Task Configuration. Each entry in the task configuration tree consists of a symbol (📄) and the program name.

With '**Insert Program Call**', the new program call is inserted in front of the cursor, and with '**Append Program Call**', the program call is appended to the end of the existing list.

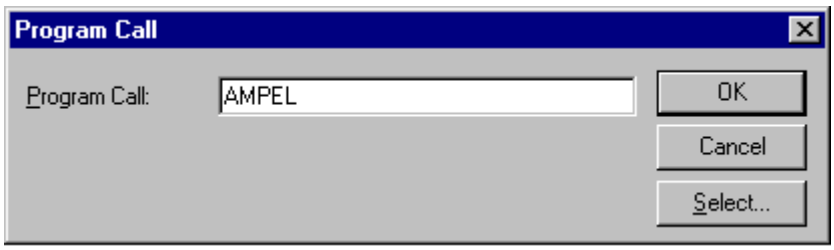


Image 6.28: Dialog box for Program Call Entry

In the field, specify a valid program name for your project, or open the Input Assistant with the **Select** button to select a valid program name. If the selected program requires input variables, then enter these in their usual form and of the declared type (for example, prg(invar:=17)).

System Events

Instead of a "task" also a "system event" can be used to call a POU of your project. The available system events are target specific (definition in target file). The list of the standard events of the target may be extended by customer specific events. Possible events are for instance: Stop, Start, Online Change.

The assignment of system events to POUs is also done in the Task configuration editor. Use the dialog 'Events', which will be opened as soon as the entry "⚡ System-events" is selected in the task configuration tree:

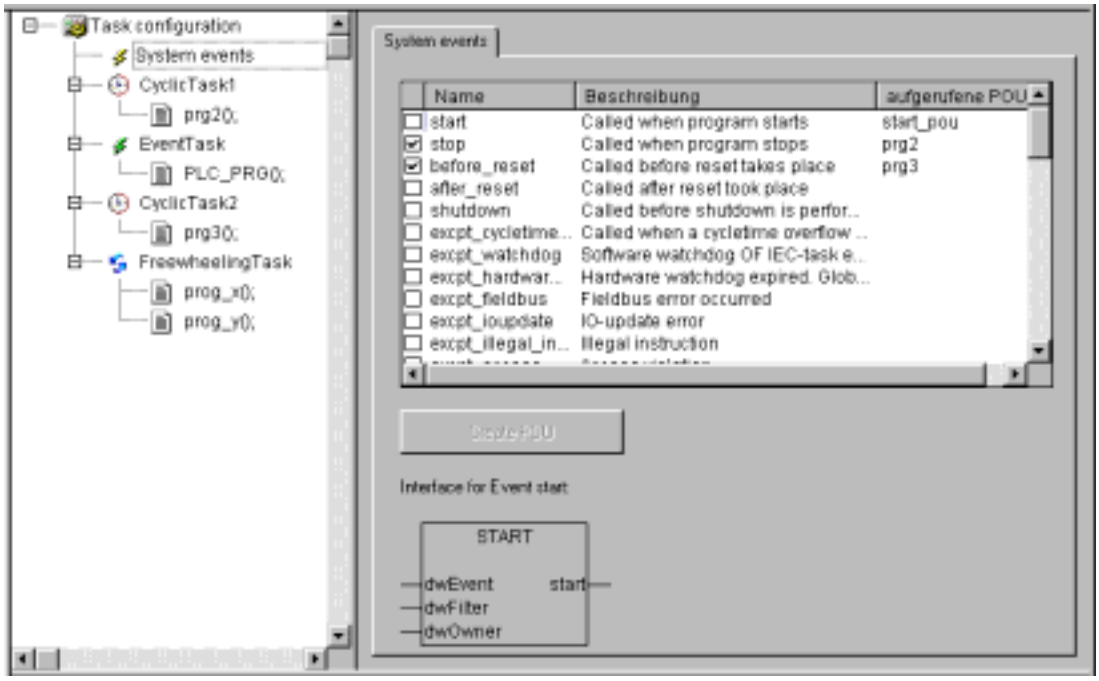


Image 6.29: Table for Assigning POUs to System Events

Each event is represented in a line: **Name** and **Description** are displayed as defined in the target file, in the column **called POU** you can enter the name of the project POU which should be called and processed as soon as the event occurs.

For this use the input assistant (<F2>) or enter manually the name of an already existing POU (e.g. "PLC_PRG" or "PRG.ACT1"), or insert a name for a not yet existing POU. In order to get this POU created in the project, press button **Create POU**. Hereupon the POU will be inserted in the Object Organizer. The input and output parameters which are required by the event will automatically be defined in the declaration part of the POU. Below the assignment table the currently selected event is displayed in a picture, showing the required parameters.

If you actually want the POU to be called by the event, activate the entry in the assignment table (☑). **Activating/deactivating** is done by a mouseclick on the control box.

Which task is being processed?

- For the execution, the following rules apply:
- That task is executed, whose condition has been met; i.e., if its specified time has expired, or after its condition (event) variable exhibits a rising edge.
- If several tasks have a valid requirement, then the task with the highest priority will be executed.
- If several tasks have valid conditions and equivalent priorities, then the task that has had the longest waiting time will be executed first.
- The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

Taskconfiguration in Online Mode

In online mode the status and number of passed through cycles of each task will be displayed in the configuration tree. The time flow is monitored in a diagram.

Precondition: the libraries **SysTaskInfo.lib** and **SysTime.lib** must be included in the project to provide functions for the internal evaluation of the task times. The libraries will be included automatically as soon as a target is set which supports the task monitoring.

Display of task status in the configuration tree:

In online mode the current status of a task will be displayed in brackets at the end of the task entry line in the configuration tree, also the number of already passed through process cycles. This update interval is the same as usual for the monitoring of PLC values. The possible stati:

Idle	has not been started since last update; especially used for event tasks
Running	has been started at least once since last update
Stop	stopped
Stop on BP	stopped, because breakpoint in task is reached
Stop on Error	Error, e.g. division by zero, page fault etc.
Stop Watchdog	cycle time has been exceeded

The task entry will be displayed red coloured in case of status 'Stop on Error' or 'Stop Watchdog' .

Display of the time flow of the tasks

If the entry 'Taskconfiguration' is selected in the configuration tree, the utilization of the tasks will be displayed in bar charts in the right part of the window:

Image 6.30: Display of the Task Execution in Online Mode

For each task a bar chart is displayed. The length of the bar represents the length of a cycle period. Below the bar as well as by appropriate marks on the bar the following measurement values are illustrated:

Min:	minimum measured runtime in µs
Akt:	last measured runtime in µs

Max:	maximum measured runtime in µs
Zyklus:	total length of a cycle in µs
Jitter:	maximum measured jitter in µs

The button **Reset** can be used to set back the values of Min., Max. and Jitter to 0.

The scaling of the chart (microseconds per Pixel) can be adjusted by the aid of a selection list at **Scaling [µs/Pixel]**.

See in the following additional commands available during online mode:

‘Extras‘ ‘Set Debug Task‘

With this command a debugging task can be set in Online mode in the Task Configuration. The text [DEBUG] will appear after the set task.

The debugging capabilities apply, then, only to this task. In other words, the program only stops at a breakpoint if the program is gone through by the set task.

'Extras' 'Callstack'‘‘

If the program is stopped at a breakpoint during debugging, then this command can be used to show the callstack of the corresponding POU. For this purpose the debug task must be selected in the task configuration tree of. The window 'Callstack of task <task name>' will open. There you get the name of the POU and the breakpoint position (e.g. "prog_x (2)" for line 2 of POU prog_x) . Below the complete call stack is shown in backward order. If you press button 'Go To', the focus will jump to that position in the POU which is currently marked in the callstack.

6.8 Sampling Trace

Sample tracing means that the progression of values for variables is traced over a certain time frame. These values are written in a ring buffer (trace buffer). If the memory is full, then the "oldest" values from the start of the memory will be overwritten. As a maximum, 20 variables can be traced at the same time. A maximum of 500 values can be traced per variable.

Since the size of the trace buffer in the PLC has a fixed value, in the event of very many or very wide variables (DWORD), fewer than 500 values can be traced.

Example: if 10 WORD variables are traced and if the memory in the PLC is 5000 bytes long, then, for every variable, 250 values can be traced.

In order to be able to perform a trace, open the object for **Sampling Trace** in the **Resources** register card in the Object Organizer. After this, you must enter the trace variables to be traced. (See **'Extras' 'Trace Configuration'**.) After you have sent the configuration with '**Save Trace**' to the PLC and have started the trace in the PLC (**'Start Trace'**), then the values of the variables will be traced. With **'Read Trace'**, the final traced values will be read out and displayed graphically as curves.

‘Extras‘ ‘Trace Configuration‘

With this command you will be given the dialog box for entering the variables to be traced, as well as diverse trace parameters for the Sampling Trace. The dialog can also be opened by a double click in the grey area of the dialog Sampling Trace.

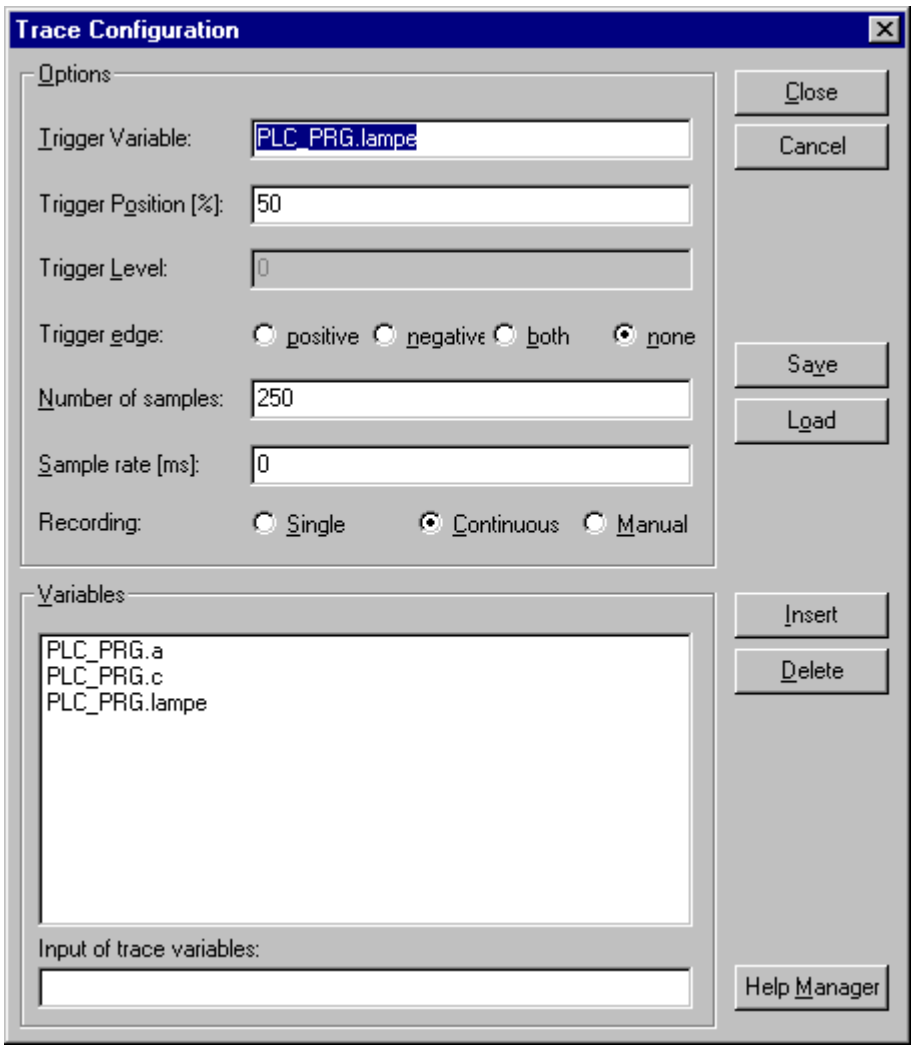


Image 6.31: Dialog Box for Trace Configuration

The list of the **Variables** to be traced is initially empty. In order to append a variable the variable must be entered in the field under the list. Following this, you can use the **Add** button or the <Enter> to append the variable to the list. You can also use the Input Assistant.

A variable is deleted from the list when you select the variable and then press the **Delete** button.

A Boolean or analogue variable can be entered into the field **Trigger Variable**. The input assistance can also be used here. The trigger variable describes the termination condition of the trace. In **Trigger Level** you enter the level of an analogue trigger variable at which the trigger event occurs. When **Trigger edge positive** is selected the trigger event occurs after an ascending edge of the Boolean trigger variable or when an analogue variable has passed through the trigger level from below to above. **Negative** causes triggering after a descending edge or when an analogue variable went from above to below. **Both** causes triggering for both descending and ascending

edges or by a positive or negative pass, whereas **none** does not initiate a triggering event at all. **Trigger Position** is used to set the percentage of the measured value which will be recorded before the trigger event occurs. If, for example, you enter 25 here then 25 % of the measured values are shown before the triggering event and 75% afterwards and then the trace is terminated. The field **Sample Rate** is used set the time period between two recordings in milliseconds. The default value ”0” means one scanning procedure per cycle.

Select the mode for recalling the recorded values: With **Single** the **Number of** the defined **samples** are displayed one time. With **Continuous** the reading of the recording of the defined number of measured values is initiated anew each time. If, for example, you enter the number ‘35’ the first display contains the first measured values 1 to 35 and the recording of the next 35 measured values (36-70) will then be automatically read, etc.. **Manual** selection is used to read the trace recordings specifically with **’Extras‘ ‘Read trace'**.

The recall mode functions independently of whether a trigger variable is set or not. If no trigger variable is set the trace buffer will be filled with the defined number of measured values and the buffer contents will be read and displayed on recall.

The button **Save** is used to store the trace configuration which has been created in a file. The standard window ”File save as” is opened for this purpose.

Stored trace configurations can be retrieved using the button **Load**. The standard window ”File open” is opened for this purpose.

Note: Please note that **Save** and **Load** in the configuration dialog only relates to the configuration, not to the values of a trace recording (in contrast to the menu commands ‘Extras’ ‘Save trace’ and ‘Extras’ ‘Load trace’).

If the field Trigger Variable is empty, the trace recording will run endlessly and can be stopped by 'Extras' 'Stop Trace'.

‘Extra‘ ‘Start Trace‘

Symbol:

With this command the trace configuration is transferred to the PLC and the trace sampling is started in the PLC.

‘Extra‘ ‘Read Trace‘

Symbol:

With this command the present trace buffer is read from the PLC, and the values of the selected variables are displayed.

‘Extra‘ ‘Auto Read‘

With this command the present trace buffer is read automatically from the PLC, and the values are continuously displayed.

If the trace buffer is automatically read, then a check (✓) is located before the menu item.

‘Extra‘ ‘Stop Trace‘

Symbol:

This command stops the Sampling Trace in the PLC.

Selection of the Variables to be Displayed

The comboboxes to the right, next to the window for displaying curves trace variables defined in the trace configuration. If a variable is selected from the list, then after the trace buffer has been read the variable will be displayed in the corresponding color (Var 0 green, etc.). Variables can also be selected if curves are already displayed.

A maximum of up to eight variables can be observed simultaneously in the trace window.

Display of the Sampling Trace

If a trace buffer is loaded, then the values of all variables to be displayed will be read out and displayed. If no scan frequency has been set, then the X axis will be inscribed with the continuous number of the traced value. The status indicator of the trace window (first line) indicates whether the trace buffer is full and when the trace is completed.

If a value for the scan frequency was specified, then the x axis will specify the time of the traced value. The time is assigned to the "oldest" traced value. In the example, e.g., the values for the last 25 seconds are indicated.

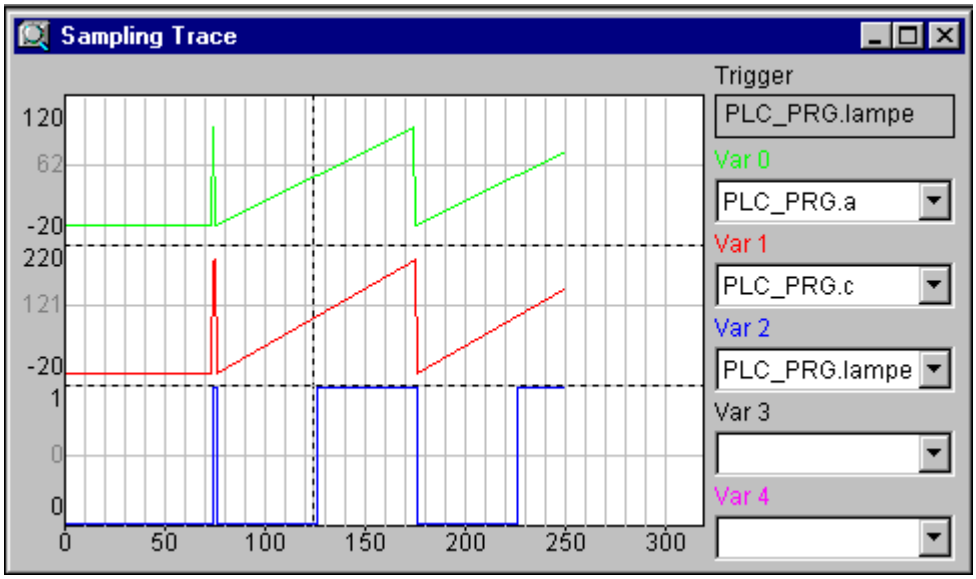


Image 6.32: Sampling Trace of Different Variables

The Y axis is inscribed with values in the appropriate data type. The scaling is laid out in a way that allows the lowest and the highest value to fit in the viewing area. In the example, Var 0 has taken on the lowest value of 6, and the highest value of 100: hence the setting of the scale at the left edge.

If the trigger requirement is met, then a vertical dotted line is displayed at the interface between the values before and after the appearance of the trigger requirement.

A memory that has been read will be preserved until you change the project or leave the system.

‘Extras‘ ‘Cursor Mode‘

The easiest way to set a cursor in the monitoring area is to click there with the left mouse button. A cursor appears and can be moved by the mouse. At the top of the monitoring window the current x-position of the cursor is displayed. In the fields next to 'Var 0', 'Var 1', ..., 'Var n' the value of the respective variable is shown.

Another way is the command 'Extras' 'Cursor mode'. With this command two vertical lines will appear in the Sampling Trace. First they are laying one on the other. One of the lines can be moved to the right or to the left by the arrow keys. By pressing <Ctrl>+<left> or <Ctrl>+<right> the speed of the movement can be increased by factor 10.

If additionally the <Shift> key is pressed, the second line can be moved, showing the difference to the first one.

‘Extras‘ ‘Multi Channel‘

With this command you can alternate between single-channel and multi-channel display of the Sampling Trace. In the event of a multi-channel display, there is a check (✓) in front of the menu item.

The multi-channel display has been preset. Here the display window is divided into as many as eight display curves. For each curve the maximum and the minimum value are displayed at the edge.

In a single-channel display, all curves are displayed with the same scaling factor and are superimposed. This can be useful when displaying curve abnormalities.

‘Extras’ ‘Show grid’

With this command you can switch on and off the grid in the graphic window. When the grid is switched on, a check (✓) will appear next to the menu item.

‘Extras‘ ‘Y Scaling‘

With this command you can change the preset Y scaling of a curve in the trace display.

In the dialog box specify the number of the desired curve (**Channel**) and the new maximum (**maximum y scale**) and the new minimum value (**minimum y scale**) on the y axis.

By doubleclicking on a curve you will also be given the dialog box. The channel and the former value are preset.

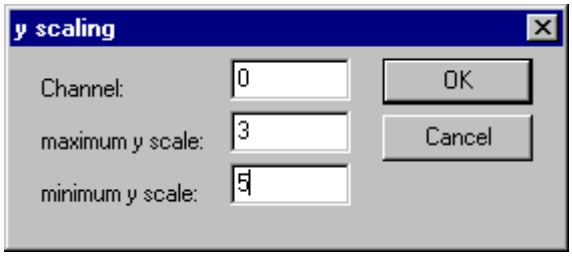


Image 6.33: Dialog Box for Setting the Y Scale

‘Extras‘ ‘Stretch‘

Symbol:

With this command you can stretch (zoom) the values of the Sampling Trace that are shown. The beginning position is set with the horizontal picture adjustment bar. With repeated stretches that follow one-after-another, the trace section displayed in the window will increasingly shrink in size.

This command is the counterpart to 'Extras' 'Compress'.

‘Extras‘ ‘Compress‘

Symbol:

With this command the values shown for the Sampling Trace are compressed; i.e., after this command you can view the progression of the trace variables within a larger time frame. A multiple execution of the command is possible.

This command is the counterpart to 'Extras' 'Stretch'.

‘Extras‘ ‘Save Trace‘

With this command you can save a Sampling Trace (values + configuration data). The dialog box for saving a file is opened. The file name receives the extension "*.trc".

Be aware, that here you save the traced values as well as the trace configuration, whereas **Save trace** in the configuration dialog only concerns the configuration data.

The saved Sampling Trace can be loaded again with 'Extras' 'Load Trace'.

‘Extras‘ ‘Load Trace‘

With this command a saved Sampling Trace (traced values + configuration data)can be reloaded. The dialog box for opening a file is opened. Select the desired file with the "*.trc" extension.

With 'Extras' 'Save Trace' you can save a Sampling Trace.

‘Extras‘ ‘Trace in ASCII-file‘

With this command you can save a Sampling Trace in an ASCII-file. The dialog box is opened for saving a file. The file name receives the extension "*.txt". The values are deposited in the file according to the following scheme:

```
CoDeSys Trace
D:\CODESYS\PROJECTS\TRAFFICSIGNAL.PRO
Cycle PLC_PRG.COUNTER PLC_PRG.LIGHT1
```

0 2 1
1 2 1
2 2 1
.....

If no frequency scan was set in the trace configuration, then the cycle is located in the first column; that means one value per cycle has been recorded at any given time. In the other respects, the entry here is for the point in time in ms at which the values of the variables have been saved since the Sampling Trace has been run.

In the subsequent columns, the corresponding values of the trace variables are saved. At any given time the values are separated from one another by a blank space.

The appertaining variable names are displayed next to one another in the third line, according to the sequence (PLC_PRG.COUNTER, PLC_PRG.LIGHT1).

6.9 Watch and Receipt Manager

Watch and Receipt Manager

With the help of the Watch and Receipt Manager you can view the values of selected variables. The Watch and Receipt Manager also makes it possible to preset the variables with definite values and transfer them as a group to the PLC ('**Write Receipt**'). In the same way, current PLC values can be read into and stored in the Watch and Receipt Manager ('**Read Receipt**'). These functions are helpful, for example, for setting and entering of control parameters.

All watch lists created ('**Insert**' '**New Watch List**') are indicated in the left column of the Watch and Receipt Manager. These lists can be selected with a mouse click or an arrow key. In the right area of the Watch and Receipt Manager the variables applicable at any given time are indicated.

In order to work with the Watch and Receipt Manager, open the object for the **Watch and Receipt Manager** in the **Resources** register card in the Object Organizer.

Watch and Receipt Manager in the Offline Mode

In *Offline Mode*, you can create several watch lists in the Watch and Receipt Manager using the '**Insert**' '**New Watch List**'.

For inputting the variables to be watched, you can call up a list of all variables with the Input Assistant, or you can enter the variables with the keyboard, according to the following notation:
<POUName>.<Variable Name>

With global variables, the POU Name is left out. You begin with a point. The variable name can, once again, contain multiple levels. Addresses can be entered directly.

Example of a multiple-level variable:
PLC_PRG.Instance1.Instance2.Structure.Componentname

Example of a global variable:

.global1.component1

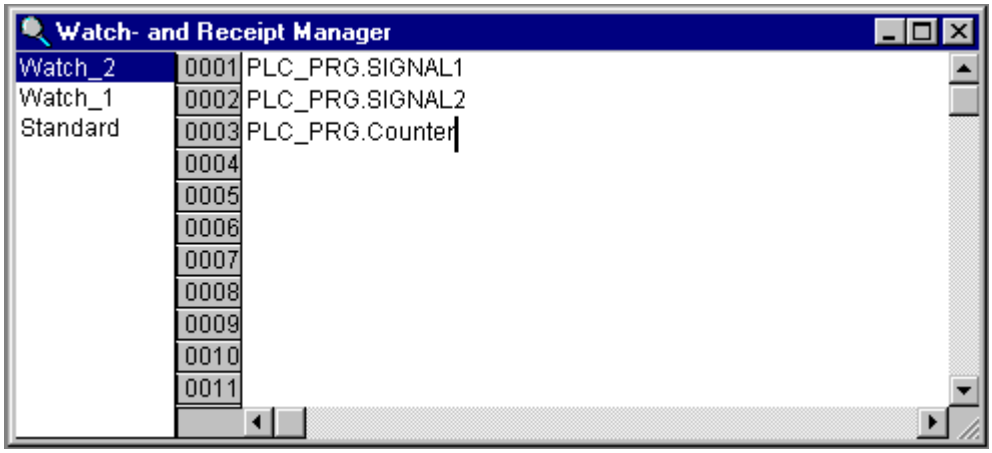


Image 6.34: Watch and Receipt Manager in the Offline Mode

The variables in the watch list can be preset with constant values. That means that in Online mode you can use the **'Extras' 'Write Receipt'** command to write these values into the variables. To do to do must use := to assign the constant value of the variable:

Example:
PLC_PRG.TIMER**:=50**

In the example, the PLC_PRG.COUNTER variable is preset with the value 6

‘Insert‘ ‘New Watch List‘

With this command a new watch list can be inserted into the Watch and Receipt Manager. Enter the desired name for the watch list in the dialog box that appears.

‘Extras‘ ‘Rename Watch List‘

With this command you can change the name of a watch list in the Watch and Receipt Manager.

In the dialog box that appears, enter the new name of the watch list.

‘Extras‘ ‘Save Watch List‘

With this command you can save a watch list. The dialog box for saving a file is opened. The file name is preset with the name of the watch list and is given the extension "*.wtc".

The saved watch list can be loaded again with 'Extras' 'Load Watch List'.

‘Extras‘ ‘Load Watch List‘

With this command you can reload a saved watch list. The dialog box is opened for opening a file. Select the desired file with the "*.wtc" extension. In the dialog box that appears, you can give the watch list a new name. The file name is preset without an extension.

With 'Extras' 'Save Watch List', you can save a watch list.

Watch and Receipt Manager in the Online Mode

In Online mode, the values of the entered variables are indicated.

Structured values (arrays, structures, or instances of function blocks) are marked by a plus sign in front of the identifier. By clicking the plus sign with the mouse or by pressing <Enter>, the variable is opened up or closed.If a function block variable is marked in the watch list, the associated context menu is expanded to include the two menu items 'Zoom' and 'Open instance'.

In order to input new variables, you can turn off the display by using the **'Extras' 'Active Monitoring'** command. After the variables have been entered, you can use the same command again to activate the display of the values.

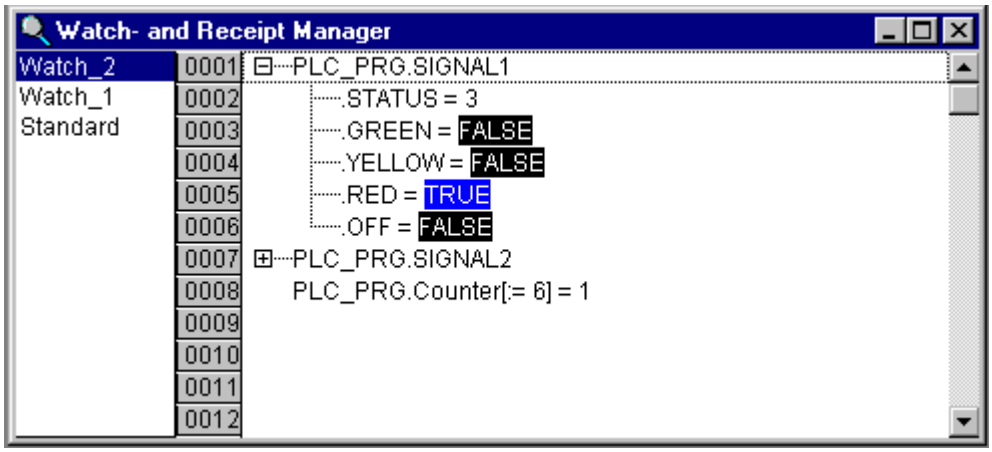


Image 6.35: Watch- and Receipt Manager in the Online Mode

In the Offline Mode you can preset variables with constant values (through inputting := <value> after the variable). In the Online Mode, these values can now be written into the variables, using the **'Extras' 'Write Receipt'** command.

With the **'Extras' 'Read Receipt'** command you can replace the presetting of the variable with the present value of the variable.

Note: Only those values the watch list are loaded which was selected in the Watch and Receipt Manager!

'Extra' 'Monitoring Active'

With this command at the Watch and Receipt Manager in the Online mode, the display is turned on or off. If the display is active, a check (✓) will appear in front of the menu item.

In order to enter new variables or to preset a value (see 'Offline Mode', the display must be turned off through the command. After the variables have been entered, you can use the same command again to activate the display of the values.

'Extras' 'Write Receipt'

With this command in the *Online Mode* of the Watch and Receipt Manager you can write the preset values (see 'Offline Mode') into the variables.

Note: Only those values of the watch list are loaded which was selected in the Watch and Receipt Manager!

‘Extras‘ ‘Read Receipt‘

With the command, in the *Online Mode* of the Watch and Receipt Manager, you can replace the presetting of the variables (see 'Offline Mode') with the present value of the variables.

Example:
PLC_PRG.Counter [:= <present value>] = <present value>

Note: Only the values of that watch list are loaded which was selected in the Watch and Receipt Manager!

Force values

In the Watch and Receipt Manager you can also **'Force values'** and '**Write values**'. If you click on the respective variable value, then a dialog box opens, in which you can enter the new value of the variable. Changed variables appear in red in the Watch and Receipt Manager.

6.10 Target Settings

The “Target Settings” is an object of the ‘Resources’. Here you define, which target shall be used for the project and how it will be configured. . If a new project is started with **'Project' 'New'**, a dialog will open where the target, that means a predefined configuration for a target, has to be set.

The list of available targets depends on which Target Support Packages (**TSP**) are installed on the computer. These describe platform specific basic configurations and also define, to which extent the configuration can be modified by the user in the CoDeSys Target settings dialogs.

Please regard: If no TSP is available, only the setting '**None**' will be offered in the target system selection box. This will switch to simulation mode automatically and no configuration settings will be possible.

Target-Support-Package

A Target Support Package (TSP) must be installed before starting by the aid of the installation program **InstallTarget** which might be part of the **CoDeSys**-Setup.

A Target Support Package (**TSP**) contains all the files and configuration information necessary to control a standard platform with a program created in **CoDeSys**. What has to be configured: codegenerator, memory layout, PLC functionality, I/O modules. In addition libraries, gateway drivers, ini-files for error messaging and PLC browser have to be linked. The central component of a TSP is one or more **Target**

files. A Target file directs to the files which are in addition necessary to configure the target. It is possible that several target files share these additional files.

The default extension for a Target file is *.**trg**, the format is binary. Additive definitions are attached to the entries in the target file which determine whether the user can see and edit the settings in the CoDeSys dialogs.

During installation of a TSP the target file for each target is put to a separate directory and the path is registered. The associated files are copied to the computer according to the information of a **Info file *.tnf** . The name of the target directory is the same as the targets name. It is recommended to store the target specific files in a directory which is named with the manufacturers name.

The files which get installed with a TSP are read when **CoDeSys** is started. The target settings which are done in the CoDeSys dialogs will be saved with the project.

Please regard:

If you use a new target file or if you have changed the existing one, **CoDeSys** has to be restarted to read the updated version.

6.10.1 Dialog Target Settings

The dialog **Target Settings** will open automatically, when a new project is created. It also can be opened by selecting the menu item ‘Target Settings’ in the register ‘Resources’ in the Object Organizer.

Choose one of the target configurations offered at **Configuration.**

If no Target Support Package has been installed, only '**None**' can be selected, which means working in simulation mode. If you choose one of the installed configurations it depends on the entries in the target files, which possibilities are left to customize this configuration in the CoDeSys dialogs. If you choose a target configuration, for which there exists no valid licence on the computer, CoDeSys asks you to choose another target.

If a configuration is selected, which is provided with the entry “HideSettings" in the corresponding target file, you only can see the name of the configuration. Otherwise there are four dialogs available to modify the given configuration:

- Target Platform
- Memory Layout
- General
- Networkfunctionality

Attention !:

Please be aware, that each modification of the predefined target configuration can cause severe changes in performance and behaviour of the target !

Press <**Default**> if you want to reset the target settings to the standard configuration given by the target file.

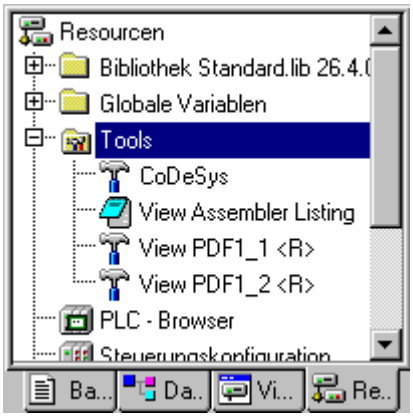
Please see Appendix 4.10 for more information about the target settings dialogs.

For information on the target settings dialogs please see keyword "Overview Target Settings. "

6.11 Tools

The object 'Tools' will be available in the 'Resources' tab if the functionality is enabled for the currently set target system. It shows all available **shortcuts** (connections) to executable files of external tools, which can be activated by a double-click in order to call these external programs from within CoDeSys. It is defined by the target file which and how many shortcuts are allowed. Depending on this definition the user can add or delete new shortcuts in the 'Tools' folder.

For example the Tools folder in the Object Organizer might look like this:



In this example four tools-shortcuts are installed. One serves for starting another CoDeSys programming sytem, one for opening the assembler listing in a text editor and two further shortcuts are available to open PDF-files. Shortcuts marked with a "<R>" cannot be modified in CoDeSys. The shortcuts may contain the connection to an editor, e.g. notepad.exe, or to a certain PDF-file, so that a doubleclick on the entry would open a notepad window showing the assembler listing respectively would open the Acrobat Reader showing the PDF-file.

Additionally you can define certain files which should be downloaded to the PLC as soon as the shortcut is activated.

Properties of available Tool
Shortcuts (Object Properties)

By a mouse-click on the plus sign at entry 'Tools' in the Resources tab of the Organizer a list of the available shortcuts will open. If you are just starting to set up a new project, you will just see those which are defined in the target file as fix entries. But if the Tools folder already had been modified you might find another shortcuts, added by a user in CoDeSys.

You can view the global tool properties (valid for all shortcuts listed in 'Tools') as well as the properties of single shortcuts.

1. Tool Properties:[#]

If 'Tools' is marked in the Resources tree, you will find the command 'Object Properties' in the context menu or in the menu 'Project' 'Object', which will open the dialog 'Tool Properties'.

There you get a table listing all tool shortcuts which might be used for the currently set target. The following parameters are shown: The **Id** (unique identification number) of the tool, the **Name of the shortcut** which is used to reference the shortcut in the Object Organizer, and the **Name of the executable file** of the tool. The button '**Advanced**' expands the dialog resp. closes the extension of the dialog:

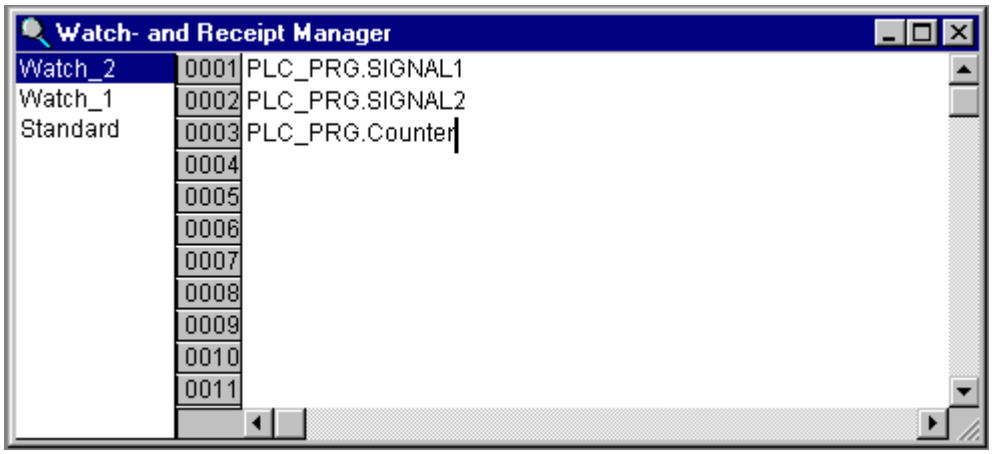


Image 6.36: Dialog 'Tool Properties'

The expanded dialog shows the global properties of the shortcut as defined in the target file. Furtheron an edit field is available where a (working) **Directory** can be defined which should be used for actions of the executable file. The path will be saved without closing the dialog as soon as you press the **Apply** button.

Properties of the Tool:

FixedCount	Number of shortcuts of the tool, which are inserted unalterably and automatically in the Tools folder. Only if "0" is entered here, the user will be able to create as many shortcuts as desired. Please regard: For shortcuts which are defined as "fix" ones by the target file, the number of possible usage in the Tools folder is predetermined and the properties cannot be modified by the CoDeSys user (cognizable by a "<R>" in the Object Organizer).
Exe-Name:	File name or full path of the executable file of the tool. If there is no entry, the file extension of the file, which is given in "Parameter Template", automatically will cause via Windows the start of the exe file of the according tool. Examples: "C:\programme\notapad.exe", "345.pdf"
DefaultDisplayName:	Name which is used to represent the tools in the Object Organizer. Possibly the template $(INSTANCE NUMBER)

[#] tools_properties

is used (see below 'Parameter Template').

Parameter Template: Templates for determining the file which should be opened by the tool. The following templates can be used, connected by the appropriate special characters:

$(PROJECT_NAME) Name of the currently opened project (File name without extension *.pro").

$(PROJECT_PATH) Path of the directory where the project file is (without indication of the drive).

$(PROJECT_DRIVE) Drive where the currently opened project is.

$(COMPILE_DIR) Compile directory of the project (including indication of the drive)

$(TOOL_EXE_NAME) Name of the exe-file of the tool.

$(DISPLAY_NAME) Name of the current shortcut, as used in the 'Tools' folder.

$(INSTANCE_NUMBER) Number of the shortcut (Instance number, running number, starting with "1")

$(CODESYS_EXE_DIR) Path of the directory where the Codesys exe-file is (including indication of the drive).

The conversion of a template you will see in the dialog for the Shortcut Properties (see below)

Example:

"$(PROJECT_NAME)_$(INSTANCE_NUMBER).cfg"
⇒ The cfg-file with the name <name of current CoDeSys project>_<shortcut number>.cfg will be opened in the tool.

DownloadFile Templates: Files, file pathes resp. templates for file which will be copied to the PLC during download.. If option Editable is activated, the list of these files will be editable in the properties dialog of the shortcut. If a file name is entered without path, the file will be searched in the directory where the codesys-exe-file is.

Example:

"a.up;$(PROJECT_NAME).zaw;$(INSTANCE_NUMBER).upp"
⇒ the files a.up, <current CoDeSys Projekt>.pro and <shortcut number>.upp will be copied to the PLC during the next download

2. Shortcut Properties:[#]

[#] shortcut_properties

Mark a shortcut entry in the 'Tools' tree in the Object Organizer and select the command 'Object Properties' in the context menu or in the 'Project' 'Object' menu. The dialog 'Shortcut Properties' will open, containing the following items:

Command	Calling the tool; paths of the exe-file and of the file which is named in 'Parameter' (predefined by the 'Parameter Template', see above) e.g.: C:\programs\notepad.exe D:\listings\textfile.txt
Parameter	Path of the file which should be called by the tool. This is defined in the target file and can be edited here, if the option 'Editable' (see below) has been activated.
Files to be downloaded	Primarily you find here the Filenames which are defined by the target file and which are also described in the Tool Properties (DownloadFileTemplate, see above). If option 'Editable' is activated in the extended dialog (see below) then you can modify the list. For this purpose press button New to open the dialog 'Filename', where you can enter another file resp. a file path. If you enter a file without path, then it will be searched in that directory, where the codesys-exe-file is. Button Delete will remove the currently marked list entry.

Button '**Standard**' resets the entries of the dialog to the default values defined by the target file.

Button '**Apply**' saves the done settings without closing the properties dialog.

Button '**Advanced**' expands the dialog so that it will look as follows :

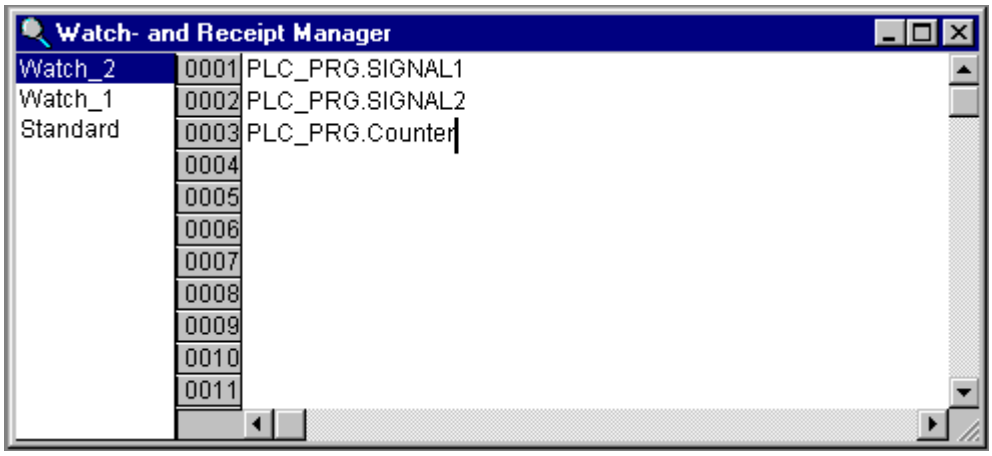


Image 6.37: Dialog 'Shortcut Properties'

Shortcut Number:	Running number, starting with 1. New shortcuts of the current tool will each get the next higher number. If a shortcut will be removed later, the numbers of the remaining shortcuts will stay unchanged. The shortcut number can be inserted in other definitions by using the template $(INSTANCE_NUMBER) (e.g. see above, 'Parameter Template').
Tool ID:	Unique identification number of the tool; defined in the target file.
Default Number of Shortcuts:	Number of shortcuts (instances) for a tool. Corresponds to the "FixedCount" defined in the target file. See above, Tool

	Properties.
Editable:	If this option is activated, it will be possible to edit the field 'Parameter' resp. the list of files which should be downloaded.

Button **OK** applies the done settings and closes the Properties dialog.

Creating new Tool Shortcuts

Select the entry 'Tools' or a shortcut entry in the Resources tree of the Object Organizer and select command 'Add Object' in the context menu or in the 'Project' 'Object' menu to open the dialog '**Create Shortcut**':

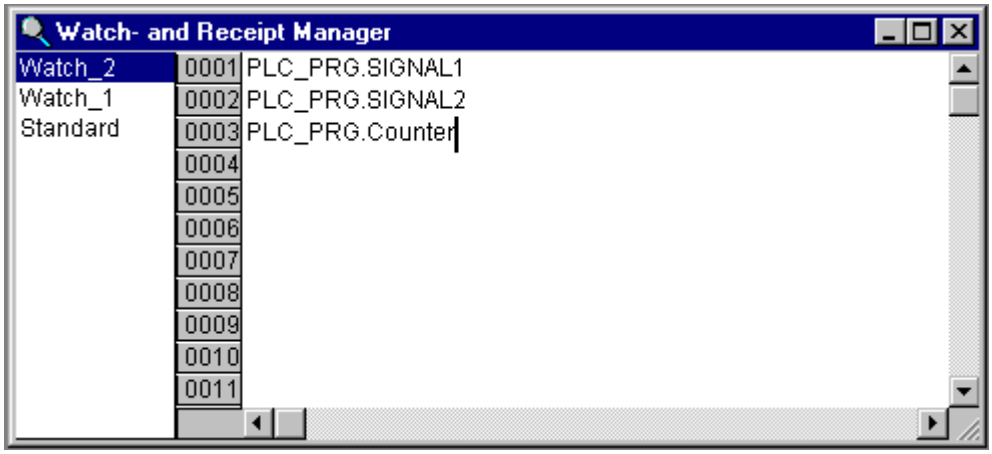


Image 6.38: Dialog 'Create Shortcut'

The table lists all tools for which new shortcuts (connections) can be created. According to the definitions in the target file the following parameters are displayed: **ID** of the tool, default **Name of the shortcut** and the **Name of the executable file**.

In order to create a(nother) shortcut for one of the offered tools, select this tool by a mouse-click in the 'ID' column. Hereupon you can modify the default name of the shortcut in the edit field **Name of the shortcut** and confirm with OK. This will only work if you enter a name which is not yet used.

OK closes the dialog and the new shortcut will be inserted in the Resources tree, represented by the shortcut name and a shortcut number which is 1 higher than the highest one used for a instance of this tool up to now.

In the area below the name field appropriate hints concerning the user inputs will be displayed.

Deleting Tool Shortcuts

To remove a shortcut from the Resources 'Tools' tree use the command 'Delete' in the context menu (right mouse button) or in the 'Project' 'Object' menu. Regard that the command will only be available if the respective tool is not defined to have a fix number of shortcuts. The shortcut numbers of the remaining shortcuts will not be changed by the removal of an entry.

Executing Tool Shortcuts

A shortcut will be executed on a double-click on the entry in the Resources tree or by the command 'Open Object' in the 'Project' 'Object' menu resp. in the context menu (right mouse button).

If the execution of the file, which is defined in the shortcut properties (Parameter), fails, then an appropriate error message will appear. If a parameter file will not be found, the exe-file of the tool will be executed and a dialog will open, asking you whether the file should be created.

If the exe-file of the tool is not found in the defined path or if no path has been defined, then the standard dialog for selecting a file will be opened and the user will be asked to enter the path of the exe-file. This path will be saved when the dialog is closed by OK and thus will be available for the tool also in other CoDeSys projects.

Saving Tool Shortcuts

When the CoDeSys project is saved, the status and settings of the 'Tools' folder in the Resources tree will also be saved.

Please note: If you save a project by 'Save as' with a new name, then you must consider the following if you use the template $(PROJECT_NAME) in the definition of the parameter file and of the files which are to be downloaded: If you had added shortcuts for a tool (FixedCount=0) in the old project, then in the new project the file names have to be renamend manually corresponding to the new project name. In contrast for a tool which is defined with a fix number of shortcuts, the template always will be replaced automatically by the current project name !

Frequently asked questions on Tools

Why do I get no entry 'Tools' in the 'Resources' ?

Only if it is defined in the target file of the currently set target system, the 'Tools' functionality will be available.

For which tools already shortcuts are available, which shortcuts can I add in the CoDeSys project ?

Open the folder 'Tools' in the 'Resources' tree of the Object Organizer by a double-click on the plus sign. You will see which tools already are connected for the current project. If you have just set up a new project and not yet worked on the Tools list, then just those entries will be displayed, which are predefined unalterably by the definitions in the target file. Otherwise you might see an already project specifically modifed tools list. In order to check, whether the list is extendable by new entries, select the command 'Add Object'. You will get a dialog offering all tools for which additional shortcuts can be created.

Which global properties do the available tools have ?

Mark the entry 'Tools' in the Object Organizer and choose the command 'Object Properties' from the context menu (right mouse button). Expand the appearing dialog by pressing the 'Advanced' button. Now you will see a list of the available tools and the corresponding parameters. Select one of the tools by a mouseclick on the ID-Symbol in order to – for example - get displayed the allowed number of shortcuts for the tool in the field 'FixedCount', or to get displayed which files will be downloaded to the PLC if the shortcut is activated. The file names or paths might be shown in the form of templates, which will be interpreted for each single shortcut as described in the following paragraph:

Which individual properties have the available shortcuts ?

Mark one of the entries below 'Tools' in the Object Organizer and select the command 'Object Properties' in the context menu (right mouse button). Press button 'Advanced' to get the parameters of the chosen shortcut. Partially they will correspond to the above described global tool properties. If allowed by the definition in the target file you can edit these parameters here.

How can I create a shortcut for a tool ?

Mark the entry 'Tools' in the Object Organizer and choose the command 'Add Object' from the context menu (right mouse button). You will see a list of available tools, but only of those for which the maximum number of shortcuts (FixedCount) is not yet reached. Choose a tool and press OK. The tool will now be inserted in the Tools folder in the Object Organizer. If you want to insert it once more, then you must enter a different tool name first, which means to mark the new entry as another instance of the same tool. For example you could name the instances of the tool Toolxy "Toolxy_1", "Toolxy_2" etc.

How can I modify the parameters of a tool ?

In order to modify the parameters of a shortcut (instance of a tool connection), mark the shortcut in the Object Organizer and choose command 'Object Properties' from the context menu. It depends on the pre-definition of the tool in the target file, whether the parameters can be edited in the properties dialog. (See in the expanded dialog whether the option 'Editable' is activated. Button 'Standard' resets all edited values to the defaults.

How can I execute a tool shortcut ?

Perform a doubleclick on the shortcut entry in the Object Organizer or select the command 'Open Object' in the context menu resp. in the 'Project' menu when the entry is marked in the Object Organizer.

7 Visualization

Overview

A visualization is a **graphical representation** of the project variables which allows inputs to the PLC program in online mode via mouse and keypad. The CoDeSys **visualization editor**, which is part of the programming system provides graphic elements which can be arranged as desired and can be connected with project variables. Thereupon in online mode the look of the graphical elements will change depending on the variables values.
Simple example: In order to represent a fill level, which is calculated by the PLC program, draw a bar and connect it to the corresponding project variable, so that the length and color of the bar will show the current fill level value. Add a text field which will display the current value in a text string and a button for starting and stopping the program.

The properties of a single visualization element as well as those of the whole visualization object will be defined in appropriate **configuration** dialogs. There it is possible to set basic parameters by activating options as well as to define a dynamic parameterizing by entering project variables.

Additional special possibilities for configuring are given by the **programability** of element properties via structure variables.

Using **placeholders** in the configuration dialogs may save a lot of effort in case you want to use the same visualization object several times with different configurations.

The visualization which is created in the programming system will in many cases be used as the only user interface available for controlling and watching the associated PLC program in online mode. For this purpose it must be possible to give inputs to the program solely by activating visualization elements. To reach this you can use **special input possibilities** during the configuration of the visualization and you have the option to define special **hotkeys** for each particular visualization.

A visualization created in CoDeSys can later be used in different ways:

1. It can be made available on **CoDeSys HMI**, a special runtime system for operating the visualization in full screen mode on a PLC computer.
2. It can be made available as a **Web-Visualization**, which allows to call and operate it via the Internet (useful for remote maintenance purposes)
3. It can be made available as a **Target-Visualization**, which can be started directly on the PLC.

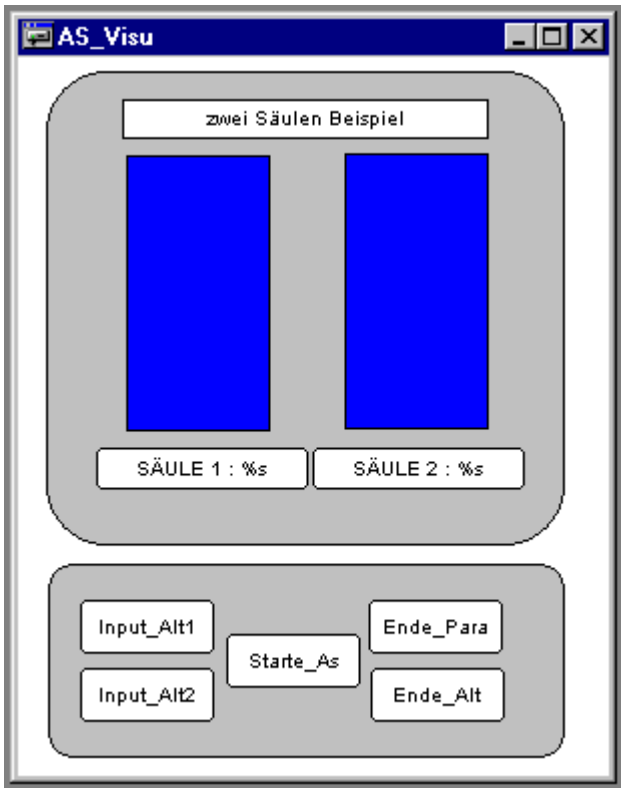


Image 7.1: Example of a Visualization

7.1 Create a new Visualization Object

A **visualization object** is a

CoDeSys object which is managed in the 'Visualization' register of the Object Organizer. It contains an arrangement of **visualization elements** and can get certain object properties. One or several visualization objects can be created in a CoDeSys project and might be linked with each other.

In order to create a visualization, you must select the register card for **Visualization** in the Object OrganizerUsing the **'Project' 'Object Add'** command, you can create a new visualization object. Open the 'New visualization' dialog, in which you can enter the name of the new visualization. Once a valid entry is made, that is not a name that is already in use and no special characters used, you can close the dialog with **OK**. A window opens, in which you can edit the new visualization.

7.1.1 Insert Visualization Elements

A visualization element is a

graphical element, which is used to fill a visualization object. The available elements are offered in the CoDeSys menu bar. Each element gets a separate configuration.

You can insert four different geometric forms, as well as bitmaps, buttons and existing visualizations, into your visualization.

Geometric forms at your disposal *include:* rectangles, rounded rectangles, ellipses/circles, and polygons.

Go to the 'Insert' menu item and select freely from the following commands: 'Rectangle', 'Rounded Rectangle', 'Ellipse', 'Polygon', 'Line‘, 'Curve', 'Bitmap', 'Visualization', 'Button', **'Metafile' (*.wmf)** .

A check appears in front of the selected command. You can also use the tool bar. The button for the selected element appears pushed down (e.g.).

If you now go to the editor window with the mouse, you will see that the mouse pointer is identified with the corresponding symbol (z.B.). Click on the desired starting point of your element and move the pointer with pressed left mouse key until the element has the desired dimensions.

If you want to create a polygon or a line, first click with the mouse on the position of the first corner of the polygon resp. on the starting point of the line, and then click on the further desired corner points. By doubleclicking on the last corner point you will close the polygon and it will be completely drawn respectively the line will be completed. If you want to create a curve (Bezier curves) determine the initial and two other points with mouse clicks to define the circumscribing rectangle. An arc is drawn after the third mouse click. You can then change the position of the end point of the arc by moving the mouse and can then end the process with a double click or add another arc with additional mouse clicks.

Furthermore pay attention, to the status bar and the change from select and insert modes.

‘Insert‘ ‘Rectangle‘

Symbol:

With the command you can insert a rectangle as an element into your present visualization. (Use, see Visualization Elements, Insert)

‘Insert‘ ‘Rounded Rectangle‘

Symbol:

With the command you can insert a rectangle with rounded corners as an element in your present visualization. (Use, see Visualization Elements, Insert).

‘Insert‘ ‘Ellipse‘

Symbol:

With the command you can insert a circle or an ellipse as an element in your present visualization. (Use, see Visualization Elements, Insert).

‘Insert‘ ‘Polygon‘

Symbol:

With the command you can insert a polygon as an element in your present visualization. (Use, see Visualization Elements, Insert).

‘Insert‘ ‘Line‘

Symbol:

With the command you can insert a line as an element into your current visualization. (Use, see Visualization Elements, Insert).

‘Insert’ ‘Curve’

Symbol:

With the command you can insert a Bezier curve as an element into your current visualization. (Use, see Visualization Elements, Insert).

‘Insert‘ ‘Bitmap‘

Symbol:

With the command you can insert a bitmap as an element in your present visualization. (Use, see Visualization Elements, Insert). Whether just a link to the bitmap file should be saved or the bitmap should be inserted as an element, can be defined in the bitmap configuration dialog (see Chapter 5.2.1).

While pressing the left mouse button, bring up an area in the desired size. The dialog box is opened for opening a file. Once you have selected the desired bitmap, it will be inserted into the area brought up.

‘Insert‘ ‘Visualization‘

Symbol:

With the command you can insert an existing visualization as an element in your present visualization. (Use, see Visualization Elements, Insert).

While pressing the left mouse button, bring up an area in the desired size. A selection list of existing visualizations opens. After you have selected the desired visualization, it will be inserted in the defined area.

An inserted visualization will also be named as a reference.

‘Insert’ ‘Button’

Symbol:

This command is used to insert a button into your current visualization. (Use, see Visualization Elements, Insert).

Drag the element to the desired size with the left mouse button held down.

If a toggle variable is configured for the button it displays the state of this variable by visually displaying whether it is pressed or not pressed. Conversely, the variable is toggled by „pressing“ the button.

'Insert' 'Metafile'

Symbol:

This command is used to insert a Windows Metafile. The standard dialog for opening a file will appear, where you can select a file (extension *.wmf). After having closed the dialog with OK the file will be inserted as an element in the visualization. In this case no link to a file will be stored, like it is done with a bitmap, but the elements of the metafile will be inserted as a group (see Chapter 5.1.2, Grouping elements).

Positioning Visualization Elements

Selecting Visualization Elements

In order to select an element, click with the mouse on the element. You can also select the first element of the elements list by pressing the <Tab> key and jump to the next by each further keystroke. If you press the <Tab> key while pressing the <Shift> key, you jump backwards in the order of the elements list.

In order to select elements, which are placed one upon the other, first select the top level element by a mouseclick. Then do further mouseclicks while the <Ctrl> button is pressed, to reach the elements in the underlying levels.

In order to mark multiple elements, press and hold the <Shift> key and click the corresponding elements, one after another; or, while holding down the left mouse button, pull a window over the elements to be selected.

In order to select all the elements, use the 'Extras' 'Select All' command.

If you are in the element list (called by **'Extras' 'Element list'**), you can select the concerned element in the visualization by selecting a line.

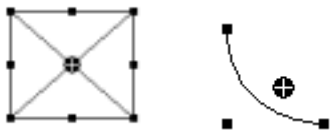
Copying Visual Elements

One or more selected elements can be inserted with the **'Edit' 'Copy'**command, the <Ctrl>+<C> key combination, or the corresponding copy symbol, and with **'Edit' 'Paste'**.

A further possibility is to select the elements and to again click in one of these elements with the **<Ctrl>** key held down. If you now hold the left mouse button down, you can separate the elements thus copied from the original.

Modifying Visualization Elements

You can select an element which has already been inserted by a mouse click on the element or by pressing the <tab> key. A small black square will appear at each corner of each of the elements, (with ellipses at the corners of the surrounding rectangle). Except in the case of polygons, lines or curves further squares appear in the middle of the element edges between the corner points.



With a selected element, the turning point (balance point) is also displayed at the same time. You can then rotate the element around this point with a set motion/angle.

The turning point is displayed as a small black circle with a white cross (⊕). You can drag the turning point with a pressed left mouse button.

You can change the size of the element by clicking on one of the black squares and, while keeping the left mouse button pressed, controlling the new outline.

With the selection of a *polygon*, you can drag each individual corner using the same technique. While doing this, if you press the <Ctrl>-key then an additional corner point will be inserted at the corner point, an additional corner point will be inserted, which can be dragged by moving the mouse. By pressing the <Shift>+<Ctrl>-key, you can remove a corner point.

Dragging Visualization Elements

One or more selected elements can be dragged by pressing the left mouse button or the arrow key.

Grouping Elements

Elements can be grouped by selecting all desired elements and performing the command 'Extras' 'Group'. The group will behave like a single element:

- the grouped elements get a collective frame; when dragging the frame all elements will be stretched or compressed; only the group can be moved to another position.
- the grouped elements get collective properties: inputs only can effect the group and not a single element. Thus the elements also get one collective configuration dialog. The property 'Change color' can not be configured for a group !

To redefine a single element of a group, the grouping must be redone by the command 'Extras' 'Break up group'. The configuration of the group will be lost in this case.

Note: As soon as you save the project as CoDeSys Version 2.1 or lower, a group of visualization elements will be resolved automatically; that means that the elements of the group will be shown as single elements in the visualization.

‘Extras‘ ‘Send to Front‘

Use this command to bring selected visualization elements to the front.

‘Extras‘ ‘Send to Back‘

Use this command to send selected visualization elements to the back.

‘Extras‘ ‘Align‘

Use this command to align selected visualization elements.

The following alignment options are available:

- **Left**: the left edge of each of the elements will be aligned to the element that is furthest to the left
- the same is true for **Right** / **Top** / **Bottom**
- **Horizontal Center**: each of the elements will be aligned to the average horizontal center of all elements

- **Vertical Center**: each of the elements will be aligned to the average vertical center of all elements

‘Extras‘ ‘Element list‘

This command opens a dialog box containing a list of all visualization elements including their **number, type** and **position**. The position is given according to the x and y position of the upper left and lower right corner of the element.

When one or more items have been selected, the corresponding elements in the visualization are marked for visual control and if necessary the display will scroll to that section of the visualization that contains the elements.

Use the **To front** button to bring selected visualization elements to the front. Use the **To behind** button to move them to the back.

Below the elements list there you find – depending on which element is currently selected - one of the following combinations of edit fields where you can modify size and position of the element:

1. If a rectangle, rounded rectangle, ellipse, bitmap, visualization, button or a meta file is currently selected, then next to the text "**Rectangle (x1, y1, x2, y2)**" there are four edit fields, where the actual x/y positions are shown and can be modified.
2. If a line, polygon or a curve is currently selected, a table will be available showing the actual **X-Position** and **Y-Position** of each of the black squares which mark the shape of the element, as soon as it is selected. These values can be edited here.

To set the modified position values in the elements list and in the visualization, press button **Set rectangle** (in case 1.) resp. **Set polygon** (in case 2.).

Use the **Delete** button to remove selected visualization elements.

Use the Undo and Redo buttons to undo or restore changes that have been made just as you would do with the commands 'Edit' 'Undo' and 'Edit' 'Redo' . In the dialog box, you can observe the changes that are being made.

Click on **OK** to close the dialog box and confirm the changes.

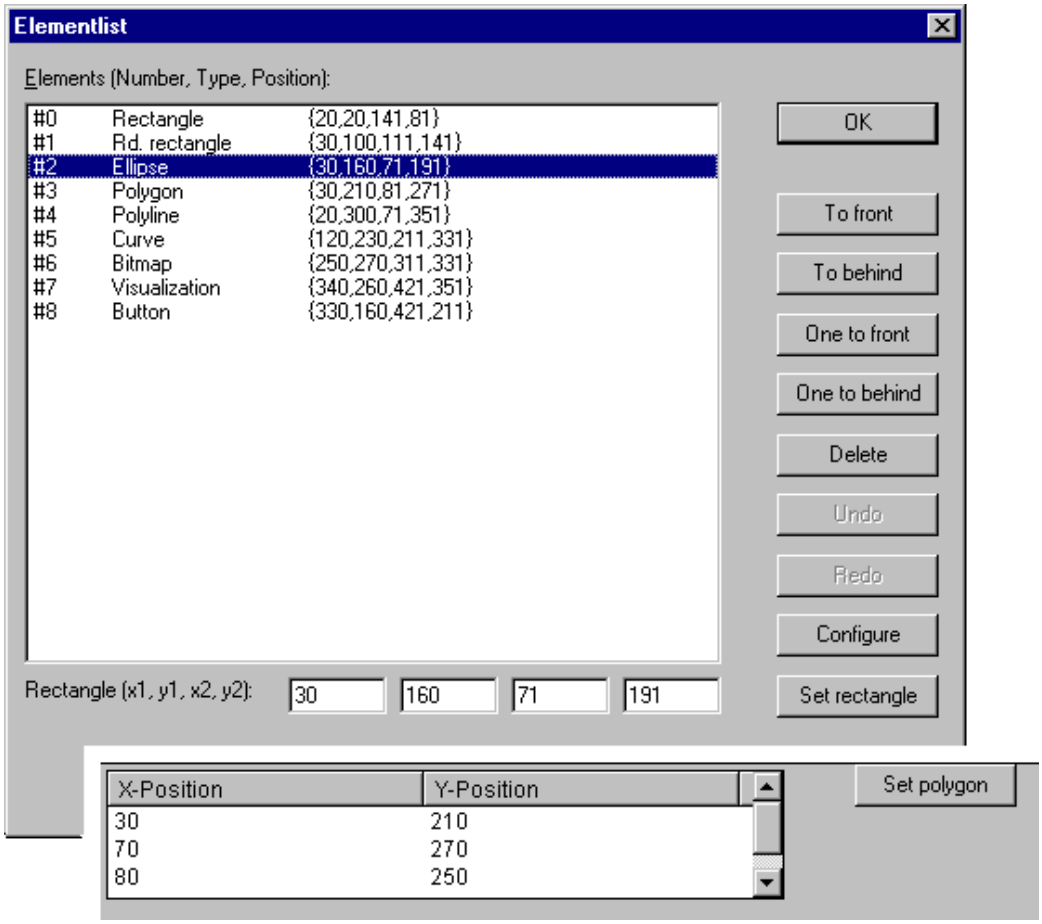


Image 7.2: Element list dialog box

Status Bar in the Visualization

If a visualization has the focus, the current **X** and **Y position** of the mouse cursor in pixels relative to the upper left corner of the image is displayed in the status bar. If the mouse pointer is located on an **Element**, or if the element is being processed, then the number of the element will be displayed. If you have selected an element to insert, then this element will also appear (for example, **Rectangle**).

7.2 Configuration of a Visualization

Overview

When configuring a visualization you have to distinguish between the configuration a particular graphic element (see Chapter 5.2.1) and the visualization object (see Chapter 5.2.2) as a whole. Correspondingly a different selection of configuration dialogs will be available, which can be opened by the command 'Configure' from Menu 'Extras' resp. from the context menu.

In this dialogs the properties of an element or object are set either by activating options or dynamically by inserting project variables. Besides that the properties can be programmed via the components of a structure variable, which can be defined for each visualization element.

Regard the order of analysis, which will be followed in online mode:

- The values which are given dynamically, i.e. via project variables, will overwrite the fix parameters defined for the same property.
- If an element property is defined by a "normal" project variable as well as by the component of a structure variable, then in online mode primarily the value of the project variable will be regarded.

Please regard the possibility of using Placeholders as well as the special input possibilities which are useful if the visualization should be used in CoDeSys HMI resp. as Target- or Web-Visualization, that means if the visualization serves as the only user interface for a PLC programm (see in the following: INTERN commands in category 'Input' of the configuration dialogs, Keyboard usage)

Placeholders

At each location in the configuration dialog at which variables or text are entered, a **placeholder** can be set in place of the respective variable or text. This makes sense if the visualization object is not to be used directly in the program, but is created to be inserted in other visualization objects as an "instance". When configuring such an **Instance**, the placeholders can be replaced with variable names or with text (see chapter 5.2.1, Visualization).

Any string enclosed in two dollar signs ($) is a valid placeholder (e.g. $variable1$, variable$x$). For each placeholder a „value group" can be defined as an input specification in the 'Placeholder list' dialog (called by 'Extras' 'Placeholder list'). With one of these values you can replace the placeholder when configuring an instance of the visualization object. A placeholder list will be available in the instance to do this replacements.

‘Extras’ ‘List of Placeholders’

This list is used at two places in CoDeSys: to manage placeholders and to configure them:

- primarily you use the list when configuring a visualization object, which later should be inserted, which means instanced, in other visualization(s). For this reason you will use placeholders instead of or additonally to variables and strings in the configuration dialogs. You can open the dialog 'Placeholders' by the command 'List of Placeholders' in the 'Extras' menu or in the context menu. The list shows three columns:

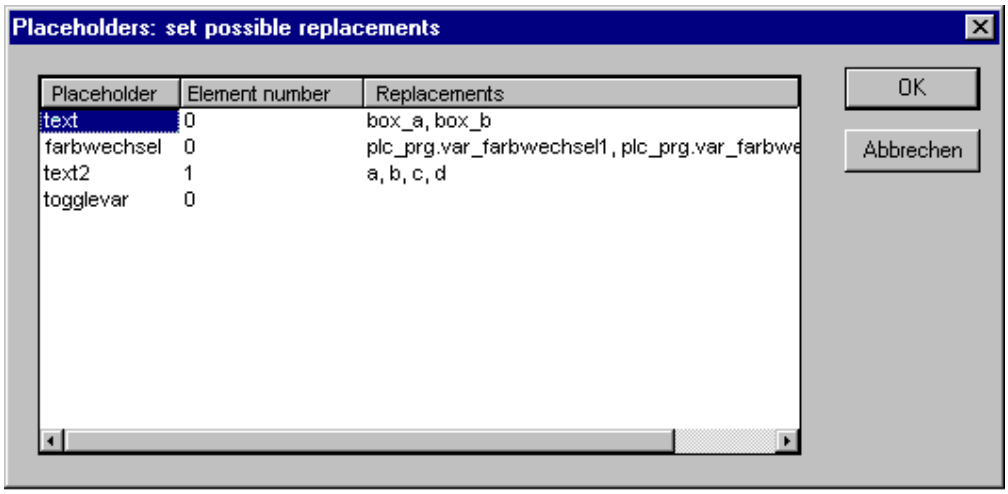


Image 7.3: Placeholder list for input of possible replacements for the placeholders

Column **Placeholder** will list all placeholders, which are currently used in the configuration of the visualization object. Column **Element number** shows the elements which contain a placeholder. In column **Replacements** you can enter one or several strings (text, variable, expression) which you want to get offered later when you will replace the placeholder during the configuration of an instance of the visualization object. If you enter more than one element, separate by commas. If you specifiy no or an impossible replacement string, then the placeholder can be replaced with any desired text later during the configuration of the visualization's reference.

- later you use the list of placeholders when **configuring an instance** of the above mentioned visualization object, that means after this object has been inserted (as a 'reference') in another visualization by the command 'Insert' 'Visualization'. For this purpose do the following to open the dialog: Select the inserted visualization, execute command 'Extras' 'Configure' and press button 'Placeholders' in Category 'Visualization'. In this case the dialog will only contain two columns:

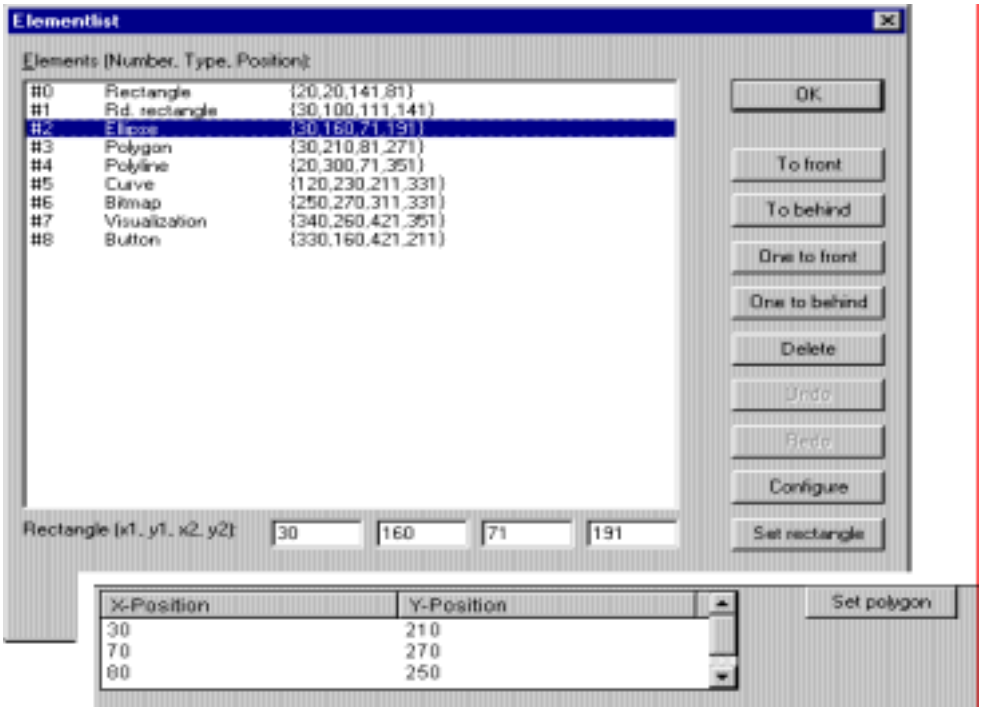


Image 7.4: List of placeholders for replacing a placeholder in a visualization instance

Column **Placeholder** – like described above – shows all placeholders which have been defined for the primary visualization object. If additionally a selection of possible replacements had been defined, this list will now be available in column 'Replacement'. Select one of the entries to replace the placeholder in the present instance. If no replacements have been pre-defined then you can manually enter an expression or variable. For this purpose perform a mouse-click on the field in column **Replacement** to open an editor field.

7.2.1 Configure Visualization Elements

‘Extras‘ ‘Configure‘

With this command, the 'Configure element' dialog opens for configuring the selected visualization element (see Select visualization element). You are given the dialog box when you doubleclick on the element.

Select a category in the left area of the dialog box, and fill out the requested information in the right area. This has to be done by activating options resp. by inserting the name of valid variables, whose values should define the property of the element.

Note: There are also configuration dialogs available for a group of elements. Regard that the settings will be valid for the "element" group. If you want to configure the particular elements of the group, you have to resolve the group.

Note: If you have defined an element property by a "static" setting as well as dynamically by a variable, then in online mode the variable will overwrite the static value (Example: "Alarm color Inside" can be defined statically in category 'Color' and additionally dynamically in category 'Colorvariables' by a variable). If the setting is controlled by a "normal" project variable as well as by a structure variable, then the value of structure variable also will be overwritten by the "normal" project variable.

Depending on the visualization element selected, various categories can be selected:

Shape	Rectangle, Rounded Rectangle, Ellipse
Text	All
Textvariables	All
Color	Rectangle, Rounded Rectangle, Ellipse, Polygon, Line, Curve; just frame color: Bitmap,Visualization
Colorvariables	Rectangle, Rounded Rectangle, Ellipse, Polygon, Line, Curve; just frame color: Bitmap,Visualization
Motion absolute	All
Motion relative	All, except Polygon, Line, Curve
Variables	All

Input	All
Tooltip	All
Security	All
Programability	All

Regard this possibility to configure a visualization element also by using a structure variable. In this case the value provided by the variable will overwrite the value which has been defined by a analogous setting in the configuration dialog !

Bitmap	Bitmap, Button
Visualization	Visualization

At locations in the element configuration where program variables are operative, the following **Entries** are possible:

- Variable names, for which input assistant is available
- Expressions which are assembled from component accesses, field accesses with constant index, variables and direct addresses.
- Operators and constants, which can be combined at will with the aforementioned expressions.
- Placeholders instead of variable names or text strings

Examples of permissible expressions:

```
x + y
100*PLC_PRG.a
TRUE
NOT PLC_PRG.b
9*sin(x + 100)+cos(y+100)
```

Function calls are not possible. Invalid expressions result in an error message on login („Invalid Watch expression...“). Examples of invalid expressions: fun(88), a := 9, RETURN.

There are two possible ways to write **global variables**: „.globvar“ and „globvar“ are equivalent. The style with a dot is not possible within an assembled expression, however.

See also below for the possible use of Placeholders.

Shape

In the visualization element configuration dialog box, you can select in the **Shape** category from among **Rectangle**, **Rounded Rectangle**, and **Ellipse** respectively **Polygon, Line** and **Curve**. The form will change into the size already set.

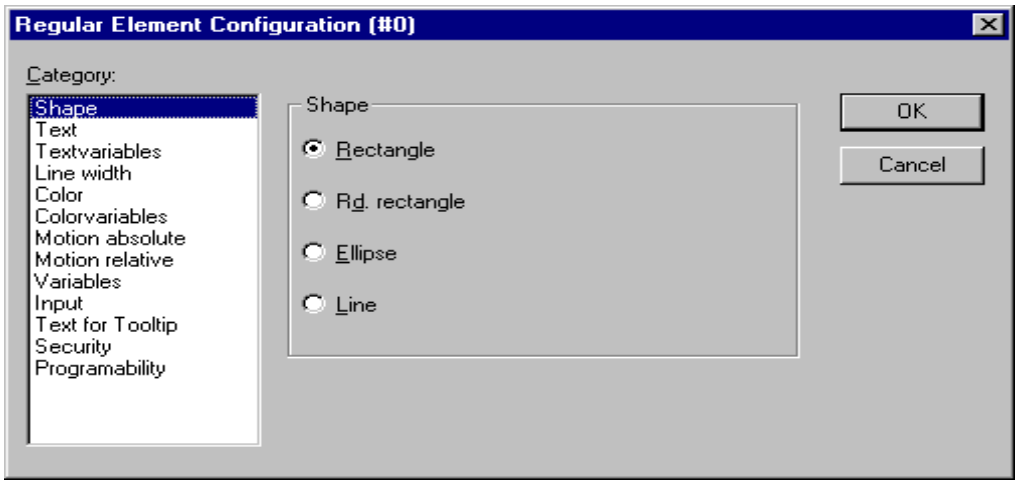


Image 7.5: Dialog Box for Configuring Visualization Elements (Shape Category)

Text

In the dialog for configuring visualization elements, you can specify a text for the element in the **Text** category. This can be entered directly or/and a variable can be defined which will determine the text string. The usage of placeholders is possible. Also the default settings for font and alignment are done here.

Enter the text in the **Content** field. With the key combination <Crtl>+<Enter> you can insert line breaks, with <Crtl>+<Tab>, tab stops. Besides the input of a pure text string you can use the following formatting sequences:

- If you include "**%s**" into the text, then this location, in Online mode, will be replaced by the value of the variable from the **Text Output** field of the **Variables** category. You also can use a formatting string, which conforms with the standard C-library function '*sprintf*': e.g. "%d" for integer types, "%f" for floating types, "%2.5f" for floating types with 2 decimal places before and 5 behind the comma, "%x" for integer types in hexadecimal format. The value of the variable will be displayed correspondingly in online mode. You can enter any IEC-conforming format strings, which fit to the type of the used variable. Attention: It is not checked whether the type which is used in the formatting string matches with the type of the variable which is defined in the 'Text Output' field !

  Example:

  Input in the 'Content' field: `Fill level %2.5f mm`
  Input in the 'Text Output' field e.g.: `fvar1  (REAL variable)`
  -> Output in online mode e.g.: `Fill level 32.8999 mm`

- If you enter "%t", followed by a certain sequence of special placeholders, then this location will be replaced in Online mode by the system time. The placeholders define the display format, see the following table:

%a	Abbreviated weekday name
%A	Full weekday name

%b	Abbreviated month name
%B	Full month name
%c	Date and time representation appropriate for locale
%d	Day of month as decimal number (01 – 31)
%H	Hour in 24-hour format (00 – 23)
%I	Hour in 12-hour format (01 – 12)
%j	Day of year as decimal number (001 – 366)
%m	Month as decimal number (01 – 12)
%M	Minute as decimal number (00 – 59)
%p	Current locale's A.M./P.M. indicator for 12-hour clock
%S	Second as decimal number (00 – 59)
%U	Week of year as decimal number, with Sunday as first day of week (00 – 53)
%w	Weekday as decimal number (0 – 6; Sunday is 0)
%W	Week of year as decimal number, with Monday as first day of week (00 – 53)
%x	Date representation for current locale
%X	Time representation for current locale
%y	Year without century, as decimal number (00 – 99)
%Y	Year with century, as decimal number
%z, %Z	Time-zone name or abbreviation; no characters if time zone is unknown
%%	Percent sign

Examples:

`%t%a %b %d.%m.%y %H:%M:%S`
-> Display in online mode: `Wed Aug 28.08.02 16:32:45`

Between the placeholders you can insert any text strings:

`%tToday is %d.%m.%y` -> Display in online mode: Today is `28.08.02`

Note: If a text string is to be transferred into a **translation file**, which will then be used in Online mode to enable switching into another national language, it must be delimited at the beginning and end by #.

Examples: "#Pump 1#" or else even "#Pump# 1"

The second case could be used in the event of multiple occurrences of the text Pump (Pump 1, Pump 2, etc.) to prevent multiple appearances in the translation.

The configured text will appear online in the prescribed alignment within the element: **horizontally left**, **center** or **right** and **vertically top**, **center** or **bottom**.

If you use the **Font** button, a dialog box for selection of the font will appear. Select the desired font and confirm the dialog with **OK**. With the **Standard-Font** button you can set the font that is selected in the project options (**'Project' 'Options' 'Editor').** If the font is changed there, then this font will be displayed in all elements except in those elements for which another font has explicitly been selected by using the **Font** button.

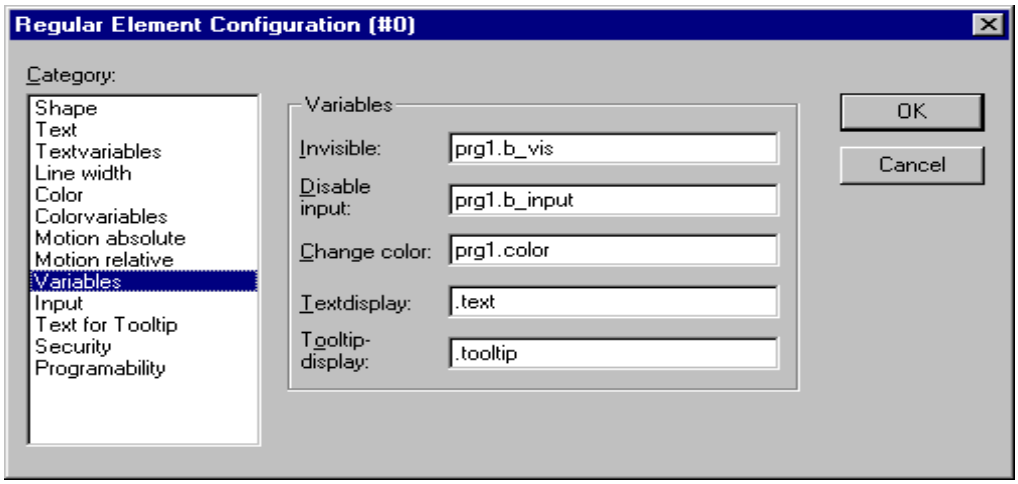


Image 7.6: Dialog Box for Configuring Visualization Elements (Text Category)

Textvariables

In category **Textvariables** of the dialog for configuring visualization elements you can specify a variable which should dynamically set color and font of that string which is defined in category 'Text'. At best enter the variable name with the aid of the input assistant (<F2>).

You can also use components of the structure *VisualObjectType* to set the text properties. For this see the description of category 'Programability'; there you will find the possible values of the particular structure components and their effect.

The parameters of the dialog:

Parameter:	**Meaning:**	**Example entry of project variable:**	**Example Usage of variable in program:**	**corresponding component of structure VisualObjectType:**
Textcolor:	Text color	"plc_prg.var_textcolor"	var_textcolor=16#FF00FF → Farbe	dwTextColor
Textflags:	Alignment (right, left, centered...)	"plc_prg.textpos"	textpos:=2 → Text right justified	dwTextFlags
Fontheight:	Font height in Pixel	".fonth"	fonth:=16; → Font height 16 pt	ntFontHeight
Fontname:	Font name	"vis1.fontn"	fontn:=arial; → Arial is used	stFontName
Fontflags:	Font display (bold, underlined, italic...)	"plc_prg.fontchar"	fontchar:=2 → Text will be displayed bold	dwFontFlags

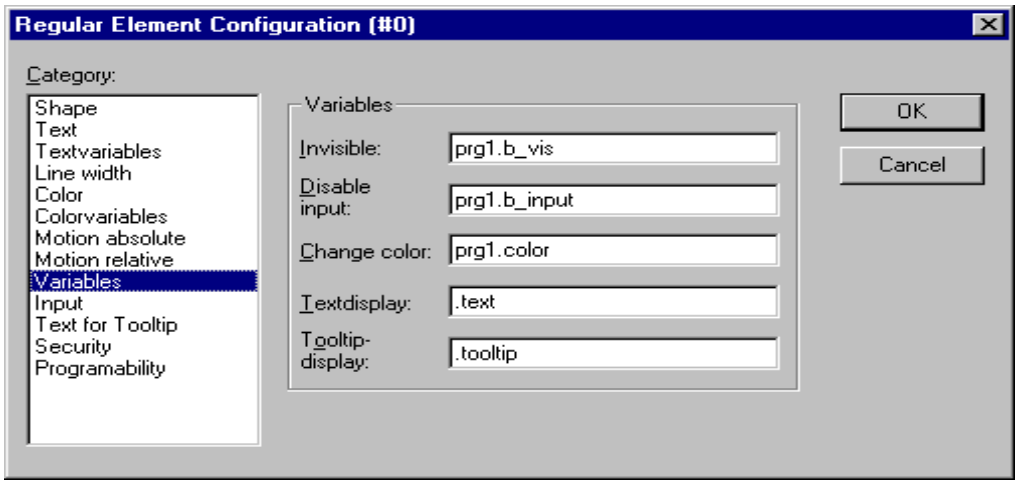


Image 7.7: Dialog for configuring visualization elements (category Textvariables)

Line width

In the dialog for configuring visualization elements, you can choose the line width for an element. As predefined options you find width settings from 1 to 5 pixel, additionally an other value can be entered manually (**Other:**), or a project variable (**Variable for line width:**) can be inserted. For the latter the input assistance ((<F2>) can be used.

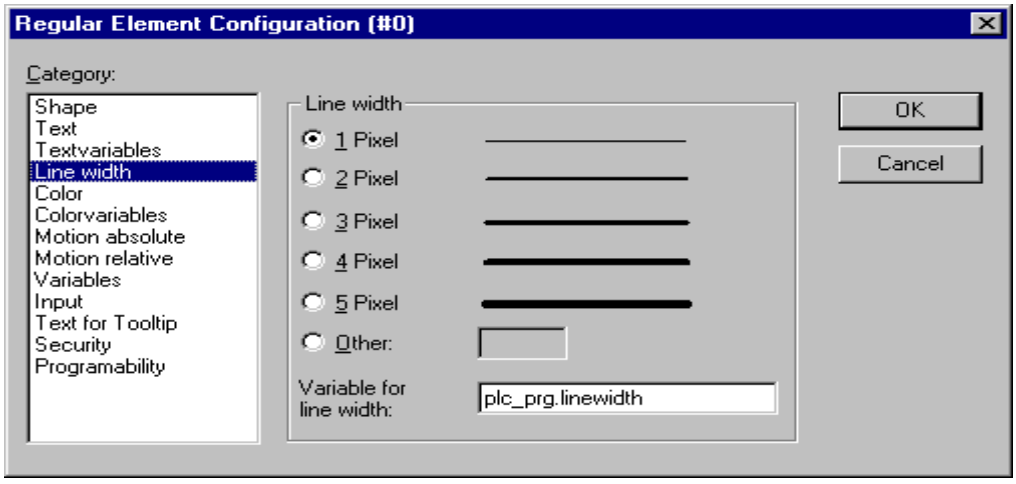


Image 7.8: Dialog Box for Configuring Visualization Elements (Line width category)

Color

In the visualization element configuration dialog box, in the **Color** category you can select primary colors and alarm colors for the inside area and for the frame of your element. Chosing the options **no color inside** and **no frame color** you can create transparent elements.

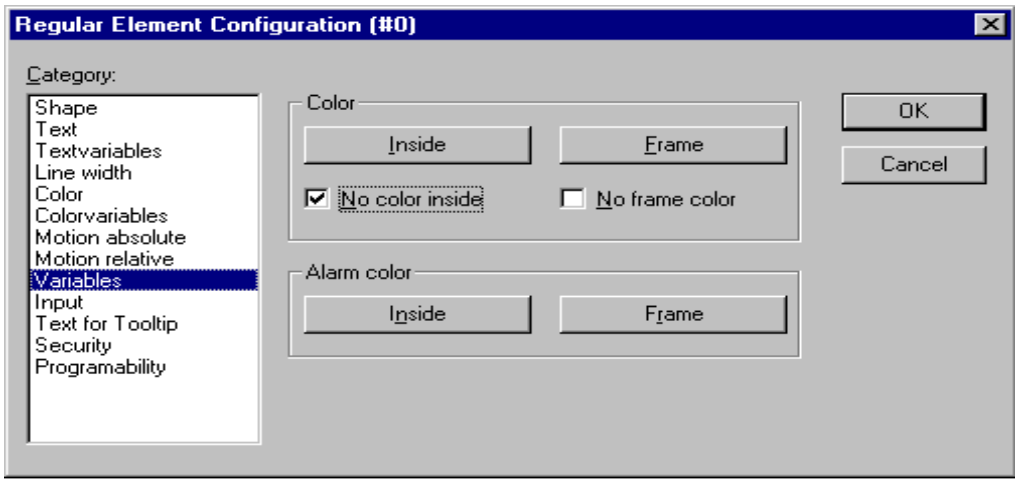


Image 7.9: Dialog Box for Configuring Visualization Elements (Color Category)

If you now enter a Boolean variable in the **Variables** category in the **Change Color** field, then the element will be displayed in the **Color** set, as long as the variable is FALSE. If the variable is TRUE, then the element will be displayed in its **Alarm Color**.

Note: The change color function only becomes active, if the PLC is in *Online Mode*!

If you want to change the color of the frame, then press the **Frame** button, instead of the **Inside** button. In either case, the dialog box will open for selection of the color.

Here can to choose the desired hue from the primary colors and the user-defined colors. By pressing the Define Colors you can change the user-defined colors.

Motion absolute

In the visualization element configuration dialog box, in the **Motion absolute** category, **X-** or **Y-Offset** fields variables can be entered. These variables can shift the element in the X or the Y direction, depending on the respective variable value. A variable in the **Scale** field changes the size of the element linear to its current value. This value, which is used as scaling factor, will be divided by 1000 implicitely, so that it is not necessary to use REAL-variables in order to get a reduction of the element. The scaling always will refer to the balance point.

A variable in the **Angle** field causes the element to turn on its turning point, depending on the value of the variable. (Positive Value = Mathematic Positive = Clockwise). The value is evaluated in degrees. With polygons, every point rotates; in other words, the polygon turns. With all other elements, the element rotates, in such a way, that the upper edge always remains on top.

The turning point appears after a single click on the element, and is displayed as a small black circle with a white cross (⊕). You can drag the turning point with a pressed left mouse button.

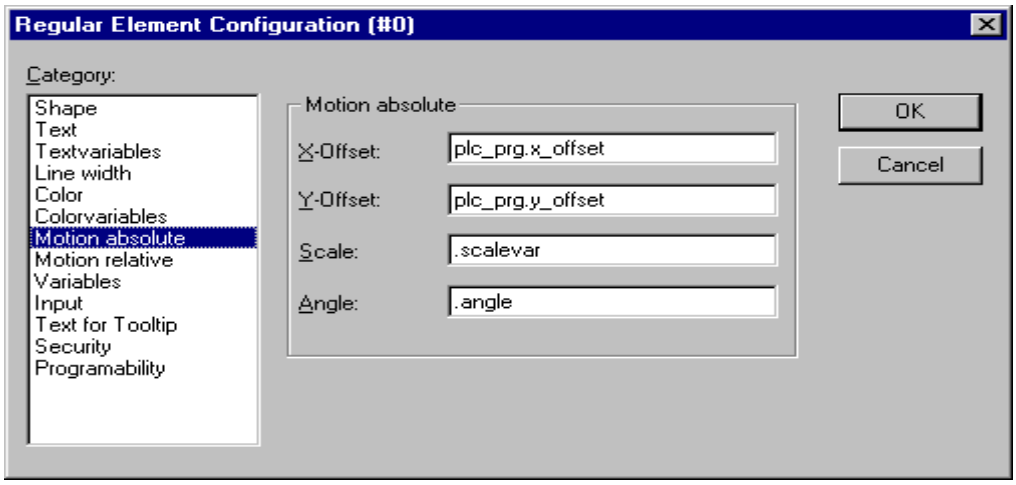


Image 7.10: Visualization Element Configuration Dialog Box (Motion Absolute Category)

Motion relative

In the dialog for configuring visualization elements in the **Motion Relative** category, you can assign variables to the individual element edges. Depending on the values of the variables, the corresponding element edges are then moved. The easiest way to enter variables into the fields is to use the Input Assistant (<F2>).

The four entries indicate the four sides of your element. The base position of the corners is always at zero. A new value in the variables, in the corresponding column, shifts the boundary in pixels around this value. Therefore, the variables that are entered ought to be INT variables.

Note: Positive values shift the horizontal edges downward, or, the vertical edges, to the right!

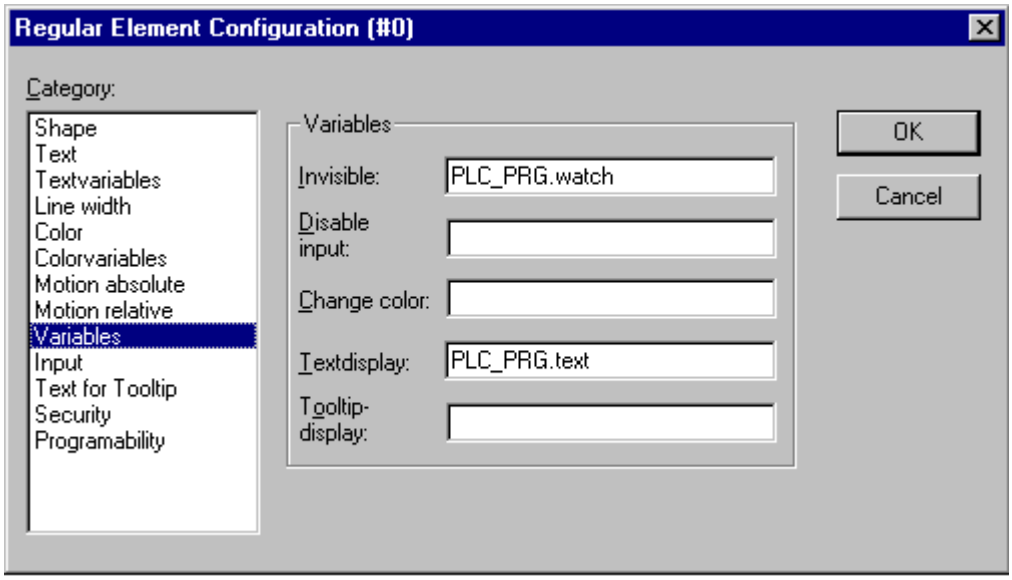


Image 7.11: Dialog Box for Configuration of Visualization Elements (Motion Relative Category)

Variables

You can enter the variables that describe the status of the visualization elements in the **Variable** category within the dialog box for configuring visualization elements. The simplest way to enter variables in the fields is to use the Input Assistant.

You can enter Boolean variables in the **Invisible** and **Change color** fields. The values in the fields determine their actions. If the variable of the **Invisible** field contains the value FALSE, the visualization element will be visible. If the variable contains the value TRUE, the element will be invisible.

Disable input: If the variable entered here is TRUE, all settings of category 'Input' will be ignored.

If the variable at the **Change color** field contains the value FALSE, the visualization element will be displayed in its default color. If the variable is TRUE, the element will be displayed in its alarm color.

Tooltip-display: Enter here a variable of type STRING whose value should be displayed in a tooltip for the element in online mode.

In the **Textdisplay** field, you can specify a variable whose value is displayed in the visualization, as long as you have entered „<name>“ in addition to the text in the **Content** field of the **Text** category. The „<name>“ is replaced in Online mode by the value of the variable from the Textdisplay field. If you want to edit the value of the variable in Online mode using the keyboard, you can do this via the 'Text input of variable' 'Textdisplay' in the Input category.

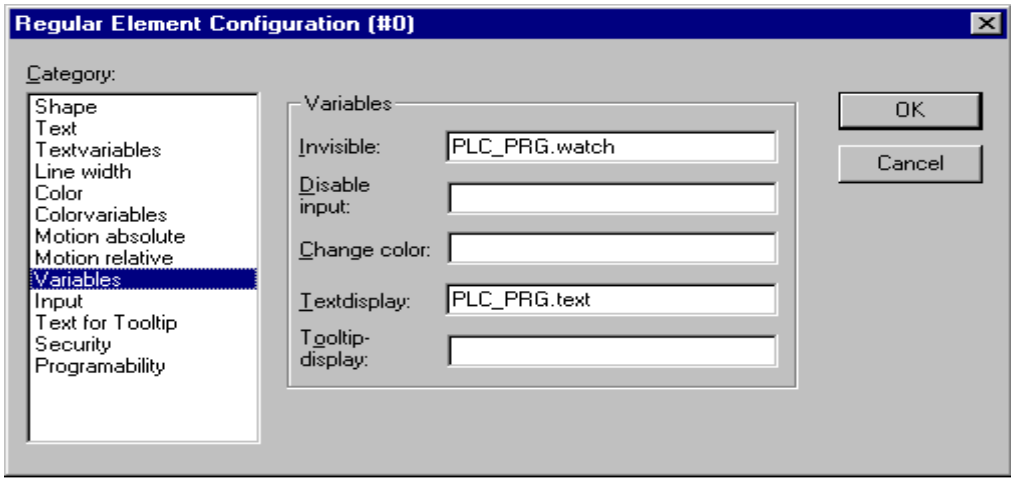


Image 7.12: Visualization Element Configuration Dialog Box (Variables Category)

Input

Selecting the field **Toggle variable** allows you, in online mode, to toggle the value of the variables which are located in the input field with every mouse click on the element. You can obtain input assistance for data entry via <F2>. The value of the Boolean variable changes with each mouse click from TRUE to FALSE and then back to TRUE again at the next mouse click, etc.

You can also use input assistance for entry of a variable for the **Tap Variable** option. This option allows you, in online mode, to change the value of the Boolean variable which is located in the input field, between TRUE and FALSE. Place the mouse cursor on the element, press the mousekey and hold it depressed. If option

Tap FALSE is activated, the value is set to FALSE as soon as the mouse key is pressed, otherwise it is set to TRUE at this moment. The variable changes back to its initial value as soon as you release the mouse key.

Selecting the field **Zoom to Vis...** allows you to enter the name of a visualization object of the same project into the edit field. You can then switch to the window showing this visualization in Online mode by clicking on the element with the mouse. The window of the target visualization will first open, followed by the closing of that of the current one. While in online mode use a mouse click to change to the element in the window of the visualization which has been entered. If a program variable of the type STRING (e.g. PLC_PRG.xxx) has been entered instead of a visualization object, then this variable can be used to define the name of the visualization object (e.g. ‚visu1') which the system should change to when a mouse click occurs (e.g. xxx:= ‚visu1).

If a visualization reference that contains **placeholders** is to be jumped to, these can be directly replaced by variable names or text when called up. For this purpose, conform to the following syntax:

<Visuname>(<Placeholder1>:=<Text1>, <Placeholder2>:=<Text2>,..., <Placeholder n>:=<Textn>)

Example:

Calling the visualization visu1, whereby the placeholders var_ref1 and var_ref2 used in visu1 are replaced by the variables PLC_PRG.var1 and PROG.var1 respectively:

```
visu1(var_ref1:=PLC_PRG.var1, var_ref2:=PROG.var1)
```

If you issue the command „ZOOMTOCALLER“ in the **Zoom to vis.** field, a backward jump into the calling visualization is achieved in Online mode by a mouse click on the element, if such a constellation was configured.

Selecting the option **Execute program** allows you to enter any executable program in the input field and then to execute it in online mode by clicking on the element with the mouse. Example: notepad C:/help.txt (the Notepad program is started and the help.txt file is opened).

Note: The configuration field **Execute program** plays a major role for the **CoDeSys operating version**, since CoDeSys program actions can be initiated here over defined commands, which are available as menu commands in the full version (see 'Special input possibilities for the CoDeSys operating version').

If you select the **Text input of variable 'Textdisplay', then** in Online mode you will get the possibility to enter an value in this visualization element which will upon pressing **<Enter>** be written to the variable that appears in the **Textdisplay** field of the **Variables** category.

Select in the scroll box which kind of input should be possible later in online mode. **Text:** An edit field will open, where you can enter the value. **Numpad** resp. **Keypad:** A window will open showing an image of the numeric resp. alphabetic keypad, where you can enter a value by activating the appropriate key elements. This might be useful if the visualization must be operatable via a touch screen. The range

of valid input values can be restricted by defining a minimum and a maximum value in the edit fields **Min:** and **Max:**.

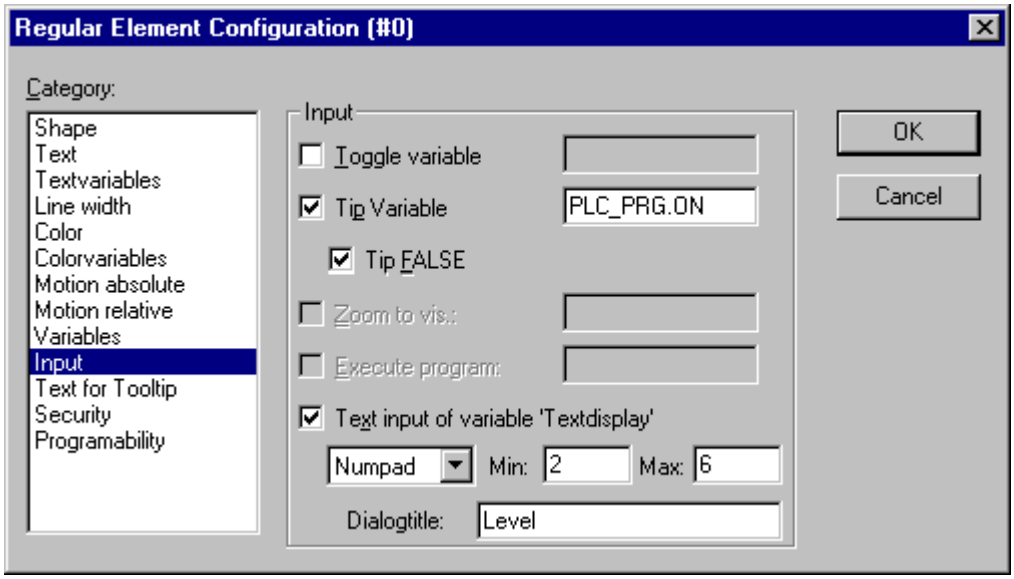


Image 7.13: Dialog to configure the visualization elements (Category Input)

Security

It might be useful that different user groups get different operating possibilities and display of a visualization. This can be reached by assigning different access rights concerning particular visualization elements. You can do this for the eight user groups which are available in CoDeSys (see also 'Project' 'Object' 'Properties' resp. 'Project' 'User Group Passwords'). The access rights can be assigned by activating the appropriate option in the configuration dialog 'Access rights' for a visualization element:

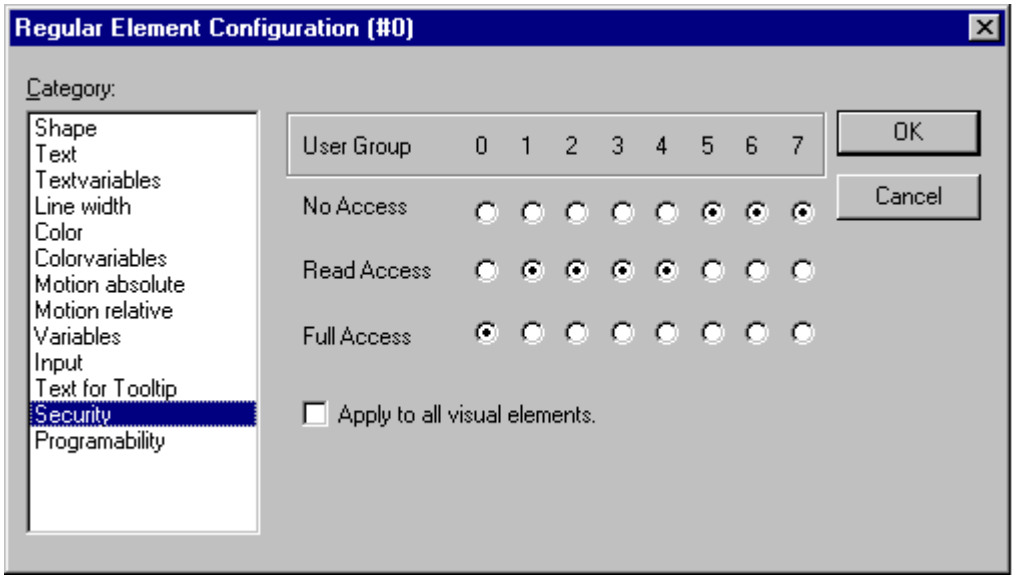


Image 7.14: Visualization Element Configuration Dialog Box (Category Security)

The access rights for a visualization element and their effect in online mode:

No Access	Element will not be visible
Read Access	Element will be visible but not operatable (no inputs allowed)
Full Access	Element is not visible and not operatable

If you want to assign the access rights also to all other elements of the visualization object, activate option **Apply** to all visual elements.

Note: Please regard, that the access rights which are set for the visualization object in the 'Project"Object' 'Properties' dialog, are independent on those of the particular visualization elements !

Programability

The properties of an element can not only be defined by a static setting or by a "normal" project variable, but also by the components of a structure variable, which is exclusively used for programming visualization elements.

For this purpose the structure **VisualObjectType** is available in the library **SysLibVisu.lib**. Its components can be used to define most of the element properties.

Note: In case of multiple definition of a element property the value of the "normal" project variables will overwrite that of the structure variable and both will overwrite a static definition.

In order to configure the element properties by using a structure variable, do the following:

Open the configuration dialog, category 'Programmability' and enter a new, unique (!) variable name in the field **Object Name:**. For this purpose you must activate the option by a mouse-click in the checkbox. The variable automatically will be declared with type **VisualObjectType**, a structure which is contained in the library **SysLibVisu.Lib**. The declaration is done implicitely and not visible for the user. Make sure that the library is included in the library manager.

After the next compile the newly assigned structure variable will be available in the project. (Hint: Activate the Intellisense functionality 'List components' in the project options, category Editor, in order to get the structure components in a selection list as soon as the variable name followed by a dot is entered).

Example: If you have defined a Object Name "visu1_line" for a visualization element, then you can program the line width of this element by e.g. "visu1_line.nLIneWidth:=4".

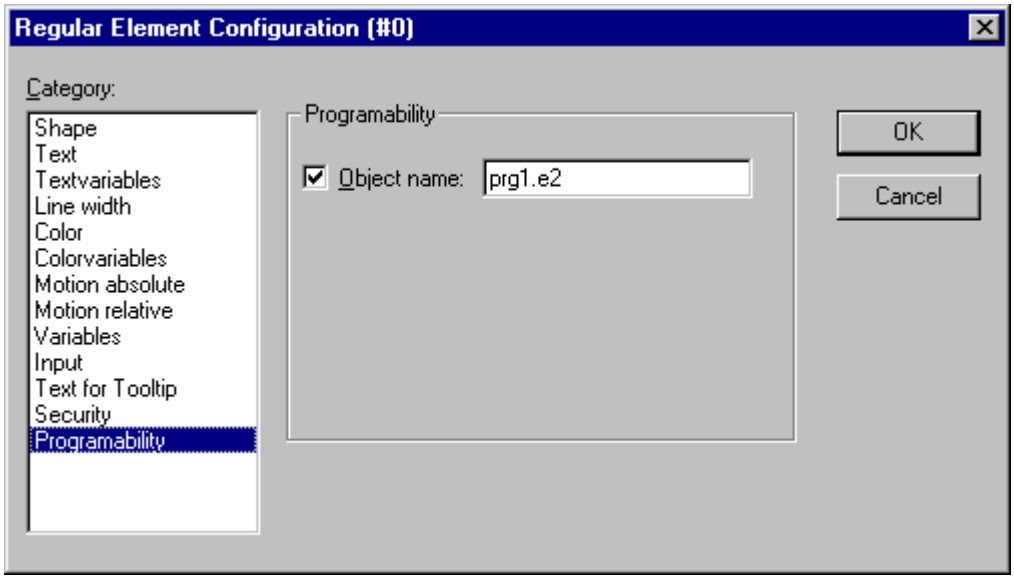


Image 7.15: Dialog for configuration of a visualization element (Category Programability

The strucure VisualObjectType:

The following table will show you all components of the structure and references to the corresponding items in the different categories of the configuration dialog:

At the beginning of the component name the data type is integrated:
n INT
dw DWORD
b BOOL
st STRING

Component (+Data type)	**Effect**	**Example (the Object Name "vis1" has been defined for the element)**	**corresponding entries in configuration dialog:**
nXOffset : INT;	Shift element in X-direction	`vis1.nXOffset:=val2;` (element is set to position X=val2)	- Cat. Motion absolute: X-Offset
nYOffset : INT;	Shift element in Y-direction	`vis1.nYOffset:=22;` (element is set to position Y=val2)	- Cat. Motion absolute: Y-Offset
nScale : INT;	Change of the size	`vis1.nScale:=plc_prg.scale_var;` (element size changes linear with change of value of plc_prg.scale_var)	- Cat. Motion absolute: Scaling
nAngle : INT;	Rotating element around its center	`vis1.anglevar:=15;` (element rotates clockwise by 15)	- Cat. Motion absolute: angle
bInvisible : BOOL;	Element is visible / invisible	`vis1.visible:=TRUE;` (element is invisible)	- Cat. Color: No color inside + No frame color - Cat. Colorvariables: Fillcolor + Framecolor
stTextDisplay : STRING;	Text is displayed in element	`vis1.TextDisplay:='ON / OFF';`	- Cat. Text: entry at Content'

		element is inscribed with this text	
bToggleColor : BOOL;	color change when toggling between TRUE and FALSE	`vis1.bToggleColor:=alarm_var;` (As soon as alarm_var gets TRUE, the element gets the color defined via the components dwFillColorAlarm, dwFrameColorAlarm resp. via the settings in category 'Colorvariables' or 'Color'.	- Cat. Input: Toggle variable + - Cat. Variables: Change color
bInputDisabled: BOOL;	if FALSE: Inputs in category 'Input' are ignored	`vis1.bInputDisabled:=FALSE;` (no input is possible for this elementt)	- Cat. Variables: 'Disable Input'
stTooltipDisplay:STRING;	Text of the tooltip	`vis1.stTooltipDisplay:='Schalter für .....';`	- Cat. Text for Tooltip: Entry in 'Content:'
dwTextFlags: DWORD;	Text position: 1 left justified 2 right justified: 4 centered horizontally 8 top 10 bottom 20 centered vertically **Note**: Always set a horizontal and a vertical position (addition of values)!	`vis1.dwTextFlags:=24;` (Text will be placed in the center of the element (4 + 20)	- Cat. Text: Horizontal and Vertical options - Cat. Textvariables: Textflags
dwTextColor : DWORD;	Text color (definition of colors see subsequent to this table)	`vis1.dwTextColor := 16#00FF0000;` (Text is blue-colored)	- Cat. Text: Font | Color - Cat. Textvariables: Textcolor
nFontHeight : INT;	Font height in Pixel. should be in range 10-96	`vis1.nFontHeight:=16;` (Text height is 16 pt)	- Cat. Text: Font | Grad' - Cat. Textvariables: Font heigth
dwFontFlags : DWORD;	Font display. Available flags: 1 italic 2 fett 4 underlined 8 canceled + combinations by adding values	`vis1.dwFontFlags:=10;` (Text is displayed blue and canceled)	- Cat. Text: Schrift | Schriftschnitt - Cat. Textvariables: Fontflags
stFontName : STRING;	Change font	`vis1.stFontName:='Arial';` (Arial is used)	- Cat. Text: Schrift | Schriftart - Cat. Textvariables: Fontname

nLineWidth : INT;	Line width of the frame (pixels)	`vis1.nLWidth:=3;` (Frame width is 3 Pixels)	- Cat. Line width
dwFillColor : DWORD;	Fill color (definition of colors see subsequent to this table)	`vis1.dwFillColor":=16#00FF0000;` (Element ist im "Normalzustand" blau)	- Cat. Color: Color | Inside - Cat. Colorvariables: Inside
dwFillColorAlarm : DWORD;	Fill color as soon as bToggleColor gets TRUE, see above) (definition of colors see subsequent to this table)	`vis1.dwFillColorAlarm:=16#00808080;` (as soon as Variable togglevar gets TRUE, the element will be displayed grey-colored)	- Cat. Color: Alarm color | Inside - Cat. Colorvariables: Inside Alarm
dwFrameColor: DWORD;	Frame color (definition of colors see subsequent to this table)	`vis1.dwFrameColor:=16#00FF0000`; (Frame is blue-colored)	- Cat. Color: Color | Frame - Cat. Colorvariables: Frame
dwFrameColorAlarm: DWORD;	Fill color as soon as bFrameColor gets TRUE, see above (definition of colors see subsequent to this table)	`vis1.dwFrameColorAlarm:=16#00808080`; (as soon as Variable vis1.bToggleColor gets TRUE, the frame will be displayed grey-colored)	- Cat. Color: Alarm color| Frame - Cat. Colorvariables: Frame Alarm
dwFillFlags: DWORD;	Color, as defined by the color variables, can be displayed or ignored 0 = show color >0 = ignore setting	`vis1.dwFillFlags:=1;` (element gets invisible)	- Cat. Color: No color inside + No frame color - Cat. Colorvariables: Fillflags
dwFrameFlags: DWORD;	Display of frame: 0 full 1 dashed (---) 2 dotted ( ) 3 dash-point (_._._) 4 dash-point-point (_.._) 8 blind out line	`vis1.FrameFlags:=1;` (Frame will be displayed as dashed line)	- Cat. Colorvariables: Frameflags

Defining color values:

Example: e1.dwFillColor := 16#00FF00FF;

A color is entered as a hex number which is composed of the Blue/Green/Red (RGB) components. The first two zeros after "16#" should be set to in each case, to fill the DWORD size. For each color value 256 (0-255) colors are available.

FF Blue component
00 Green component
FF Red component

Example for a blinking visualization element:

Define a global variable 'blink1' of type VisualObjectType in the configuration of a rectangle. In a program of function block the value of a component of the structure can be modified.

```
PROGRAM PLC_PRG
VAR
     n:INT:=0;
     bMod:BOOL:=TRUE;
END_VAR
(* Blinking element *)
n:=n+1;
bMod:= (n MOD 20) > 10;
IF bMod THEN
     blinker.nFillColor := 16#00808080;        (* Grau *)
ELSE
     blinker.nFillColor := 16#00FF0000;        (* Blau *)
END_IF
```

*Special input possibilities for**CoDeSys HMI***

CoDeSys HMI is a pure operating version, a runtime system for a CoDeSys visualization. There are no menus and status and tool bars available to the user and no possibility to modify the code. When a visualization is created with **CoDeSys** the principal control and monitoring functions in a project can be assigned to visualization elements thus making them accessible via mouse click or keyboard in Online mode. So the program can be controlled exclusevly by the visualization running in **CoDeSys HMI**.

See in the following some special input possibilities to configure visualization elements for the purpose of beeing used in CoDeSys HMI. They are available in the configuration dialog for a visualization element:

Enter internal commands in the field **Execute program** in the category **Input** according to the following syntax:

INTERN <COMMAND> [PARAMETER]*

The following table shows the available internal commands. Some of them expect to receive several parameters, which are then entered separated by spaces. Optional parameters are enclosed in square brackets. For those commands which require that a Watch list be specified, a placeholder can be used instead of the direct name.

Command	The equivalent in the full version of CoDeSys	Explanation
LANGUAGE	<visualization settings	The dialog for visualization settings which includes the category language gets openend.
LANGUAGE <language>	<visualization settings	The desired language file is chosen without using the dialog for visualization settings.
DEFINERECEIPT name	<Select watch lists>	A watch list is selected from the receipt manager which enters

		your name (name) when the command is given. The variables in this watch list are registered and displayed.
WRITERECEIPT name	‘Write receipts'	The name of a watch list of the receipt manager is expected. The receipt of this watch list will be written. A previous execution of DEFINERECEIPT is not necessary.
SAVEWATCH	‘Save watch list'	The receipt will be read into the current watch list which will be stored in a file. Important: call a previous DEFINERECEIPT to define the current receipt !
LOADWATCH	‘Load watch list' + ‘Write receipt'	The standard window ‚File open‘ appears, from which a previously stored receipt can be selected. This receipt will be immediately written into the controller system.
CHANGEUSERLEVEL	-	A dialog for setting the user group level will open. The eight CoDeSys user group levels are offered for selection.
CHANGEPASSWORD	'Project' 'User Group Passwords...'	A dialog for changing the user group password will appear.
EXITPROGRAM	‚File‘ ‘Close'	The program will be exited.
PRINT	‚File‘ ‘Print'	The current visualization will be printed out online.
HELP <name of help file>	Call of a help file	Depending on which language is set for the visualization, a help file will be called which is entered for that language in the **CoDeSys**.ini file. (see 'Extras' 'settings')
TRACE	<Open object trace recording	The window for trace recording will be opened. The menu commands Trace Start, Read, Stop, Save, Load

		which are available in the full version of CoDeSys are available in this window.

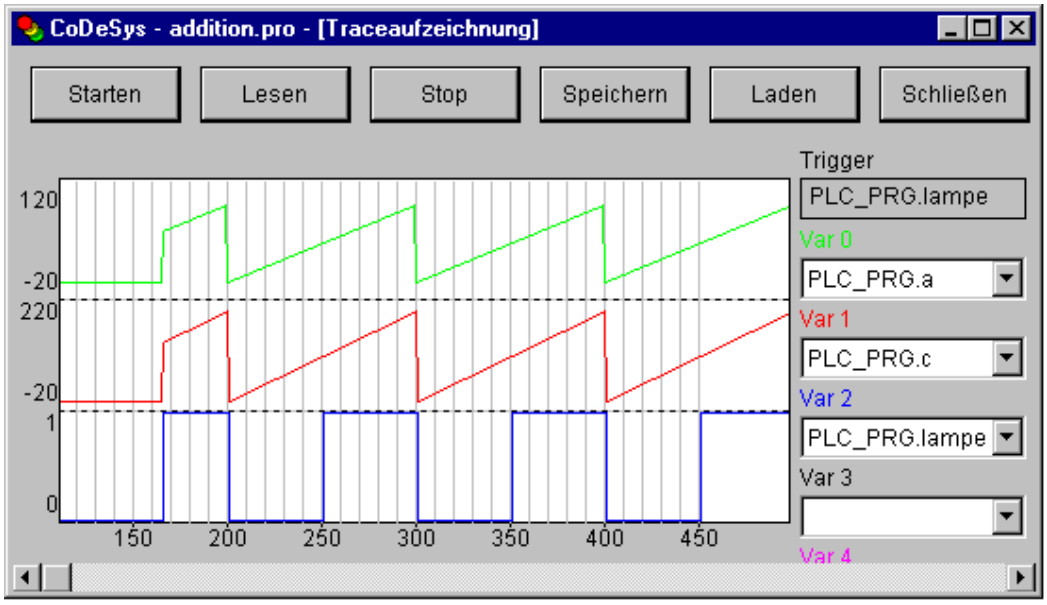


Image 7.16: Dialog for the trace recording in the operation version

ToolTip

The dialog Text for Tooltip offers an input field for text which appears in a text field as soon as the mouse cursor is passed over the object in online mode. The text can be formatted with line breaks by using the key combination <Ctrl> + <Enter>.

Bitmap

You can enter the options for a bitmap in the **Bitmap** category within the visualization element configuration dialog box.

Enter the bitmap file and its path in the **Bitmap** field. You can use the **...** button to open the standard Windows Browse dialog box from which you can select the desired bitmap.

All other entries affect the **frame** of the bitmap.

By selecting **Anisotropic**, **Isotropic** or **Fixed** you specify how the bitmap should react to changes in the size of the frame. **Anisotropic** means that the bitmap remains the same size as the frame which allows you to change the height and width of the bitmap independently. **Isotropic** means that the bitmap retains the same proportions even if the overall size is changed (i.e., the relationship between height and width is maintained). If **Fixed** is selected, the original size of the bitmap will be maintained regardless of the size of the frame.

If the **Clip** option is selected together with the **Fixed** setting, only that portion of the bitmap that is contained within the frame will be displayed.

If you select the **Draw** option, the frame will be displayed in the color selected in the **Color** and **Alarm color** buttons in the color dialog boxes. The alarm color will only be used if the variable in the **Change Color** field in the **Variable** category is TRUE.

In the selection list in the lower part of the dialog you can define whether the bitmap should be inserted in the project (Insert) or whether just a link to an external bitmap-file (path as entered above in the 'Bitmap' field) should be created (**Link to file**). It is reasonable to keep the bitmap file in the project directory, because then you can enter a relative path. Otherwise you would enter an absolute path and this might cause problems in case you want to transfer the project to another working environment.

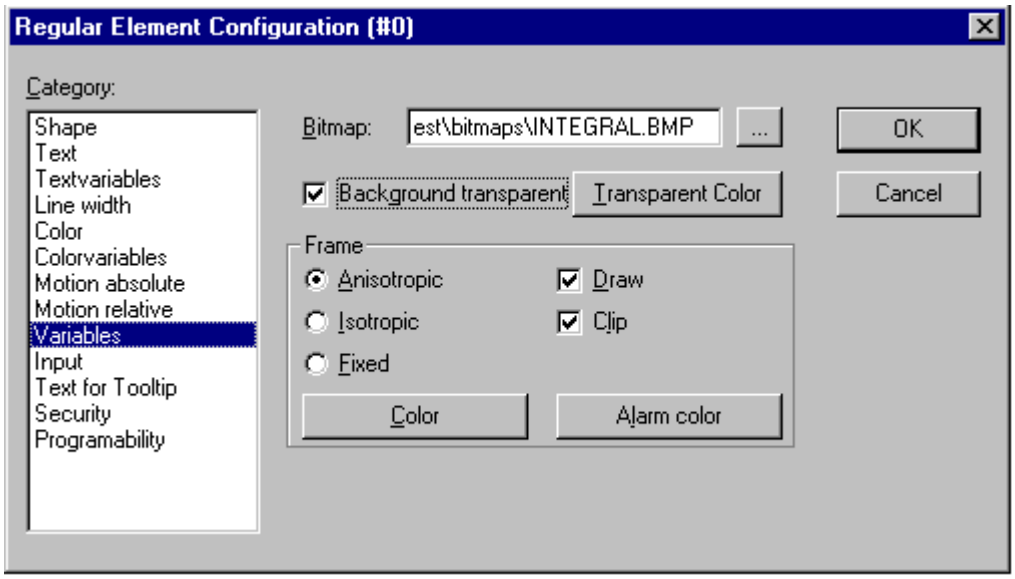


Image 7.17: Visualization Element Configuration Dialog Box (Bitmap Category)

Visualization

You can enter the options for a visualization as an element in another visualization in the **Visualization** category within the visualization element configuration dialog box. Enter the object name for the visualization in the **Visualization** field. Use the **...** button to open a dialog box containing the visualizations available in this project. Any visualization may be used with the exception of the current one.

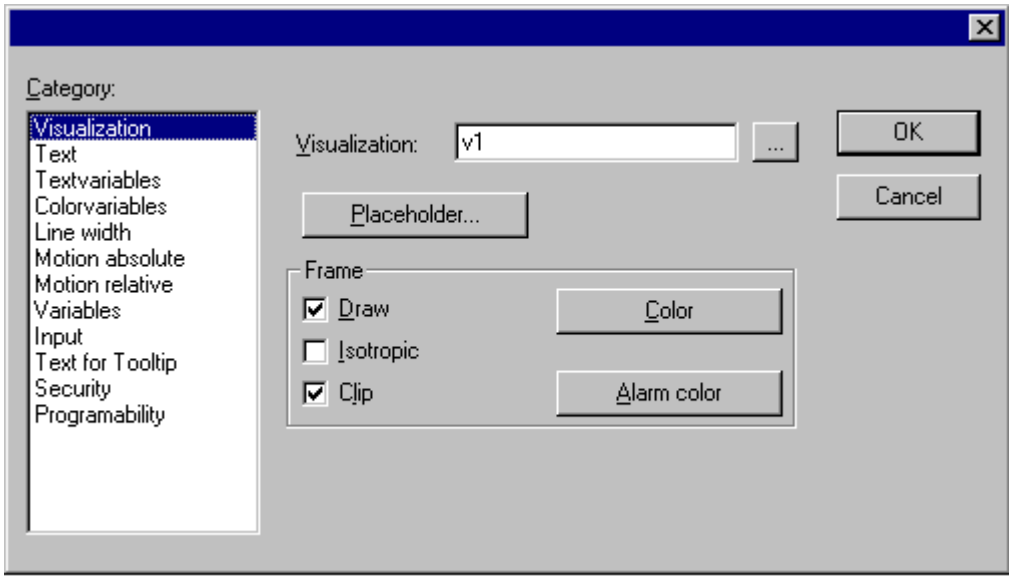


Image 7.18: Visualization Element Configuration Dialog Box (Visualization Category)

All other entries affect the visualization **frame**.

If you select the **Draw** option, the frame will be displayed in the color selected in the **Color** and **Alarm color** buttons in the color dialog boxes. The alarm color will only be used if the variable in the **Change Color** field in the **Variables** category is TRUE.

If **Isotropic** is selected, the proportions of the visualization will be maintained even if the size changes (i.e., the relationship between height and width will remain the same). Otherwise the proportions can be changed.

If the **Clip** option is selected in Online mode, only the original portion of the visualization will be displayed. For example, if an object extends beyond the original display area, it will be clipped and may disappear from view completely in the visualization.

The **Placeholder** button leads to the 'Replace placeholder' dialog. It lists in the 'Placeholder' column all the placeholders used in the inserted visualization POU and offers in the 'Replacements' column the possibility of replacing these with a definite value. Which replacements are possible in a given case depends on whether a value group was predefined in the 'Extras' 'Placeholder list' dialog. If this is the case, it will be displayed in a combo box for selection. If nothing was pre-defined, double clicking on the corresponding field in the Replacements column opens an editing field which can be filled in as desired.

A further possibility for replacing placeholders in references occurs directly during the calling of a visualization, by entry into the **Zoom to vis.** option field in the 'Input' category of the configuration dialog for a visualization element.

Note: No control of the chronological sequence of replacements is possible! Therefore no placeholders should be replaced with text that also contains placeholders!

Note: When using placeholders it is no longer possible to check for invalid entries in the configuration of the visualization element immediately upon compilation of the project. Hence the appropriate error messages are first issued in Online mode (...Invalid Watch expression..).

Example of an application of the placeholder concept:

Instances of a function block can easily be displayed with the help of references of the same visualization. For example, in configuring the visualization visu, which visualizes the variables of function block, one could begin each variable entry with the placeholder FUB (e.g. FUB.a). If a reference from visu is then used (by inserting visu in another visualization or by calling via 'Zoom to vis.'), then in the configuration of this reference the placeholder FUB can then be replaced with the name of the function block instance to be visualized.

This might look like shown in the following:

In the project define a function block containing the following declarations:

```
FUNCTION_BLOCK fu
VAR_INPUT
     changecol : BOOL;   (* should cause a color change in the
visualization *`)
END_VAR
```

In PLC_PRG define two instances of 'fu':

```
inst1_fu : fu;
inst2_fu : fu;
```

Create a visualization object 'visu'. Insert an element and open the configuration dialog, category 'Variables'. Enter in field 'Change color' the following: "FUB.changecol". Open category 'Input' and enter in field 'Tip Variable' "FUB.changecol". Open category 'Text' and enter"FUB - change color ".

Create another visualization object 'visu1'.

Insert visualization 'visu' twice in 'visu1' (two references of 'visu').

Mark the first reference of 'visu' and open the configuration dialog of category 'Visualization'. Press button 'Placeholder', so that the placeholder list will be displayed. There replace entry 'FUB' by 'PLC_PRG.inst_1'.

Now mark the second reference of 'visu' and (like described for the first one) replace 'FUB' by 'PLC_PRG.inst_2'.

Now in online mode the values of the variables which are used to configure the two instances of 'fu' will be visualized in the corresponding reference of 'visu'.

Of course the placeholder FUB can be used at all places in the configuration of 'visu' where variables or text strings are entered.

Attention: Online behaviour of a visualization reference: If you insert a visualization and then select and configurate this reference, it will be regarded as a single object and in online mode will react to inputs correspondingly to its configuration. In contrast: if you do not configurate the reference, then in online mode its particular visualization elements will react exactly like those of the original visualization.

7.2.2 Configure Visualization Objects

Besides the configuration of the individual visualization elements (see Chapter 5.2.1 also the visualization object ‘on the whole can get configured. This is possible concerning the settings for frame, language, grid, background etc. as well as the assignment of special hotkey definitions (keyboard usage), which should be valid for exactly one visualization object.

‘Extras‘ ‘Settings‘

When this command is used, a dialog box will open in which you can make certain settings that affect the visualization.

Note: The categories **Display, Frame** and **Language** also can be edited in the online mode.

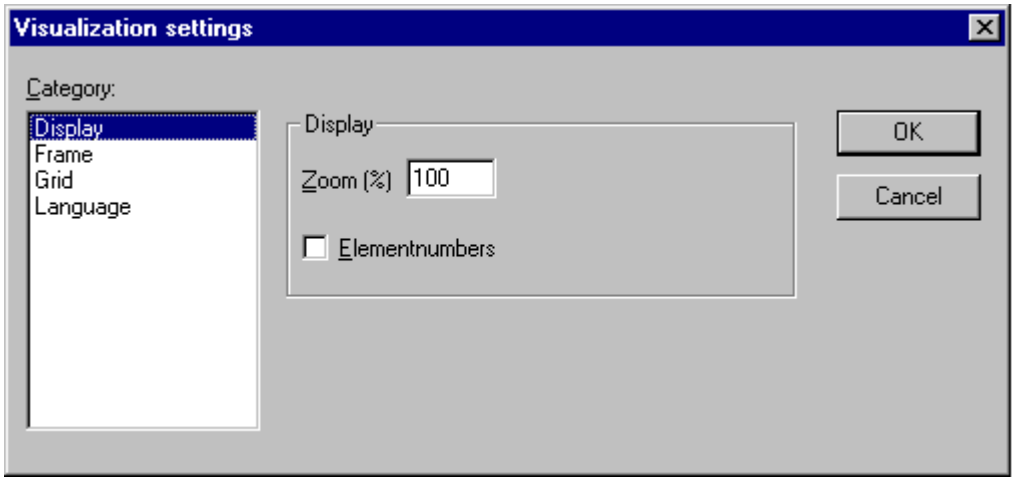


Image 7.19: Setting dialog for visualizations (Category Display)

Category **Display** : Enter a zoom factor into the field **Zoom** of between 10 and 500 % in order to increase or decrease the size of the visualization display.

Category **Frame**: If **Auto-scrolling** is selected, the visible portion of the visualization window will move automatically when you reach the edge while drawing or moving a visualization element. If **Best fit in Online mode** is selected, the entire visualization including all elements will be shown in the window in Online mode regardless of the size of the window. When **Include Background Bitmap** is selected, the background bitmap will be fitted into the window as well, otherwise only the elements will be considered.

Category **Grid:** Define here whether the grid points are **visible** in the offline mode, whereby the spacing between the visible points is at least 10 even if the entered size is smaller than that. In this case the grid points only appear with a spacing which is a multiple of the entered size. Selecting **Active** causes the elements to be placed on the snap grid points when they are drawn and moved. The spacing of the grid points is set in the field **Size**.

Category **Language** : Here you can specify in which national language the text that you assigned to an element in the **Text** and **Text for Tooltip** options should be displayed.

Note: The text display changes only in Online mode!

If different languages should be selectable for the display in online mode, either a *.tlt translation file for the project, or a *.vis language file created especially for the visualization must exist.

Regarding creating a **translation file**, please see Chapter 2.3, Manage projects, or the menu item 'Project' 'Translate into other languages'.

For creating a special *.vis **language file** for the visualization, see below. This option is retained for reasons of compatibility with projects created under Version 2.1.

In order to select a translation or language file, activate the **Language file** option in the dialog and enter in the input field next to it the appropriate file path, or obtain the help of the standard file opening dialog via the button.

In the selection window under **Language** you can now select among the options *German* and *English* in the example shown here.

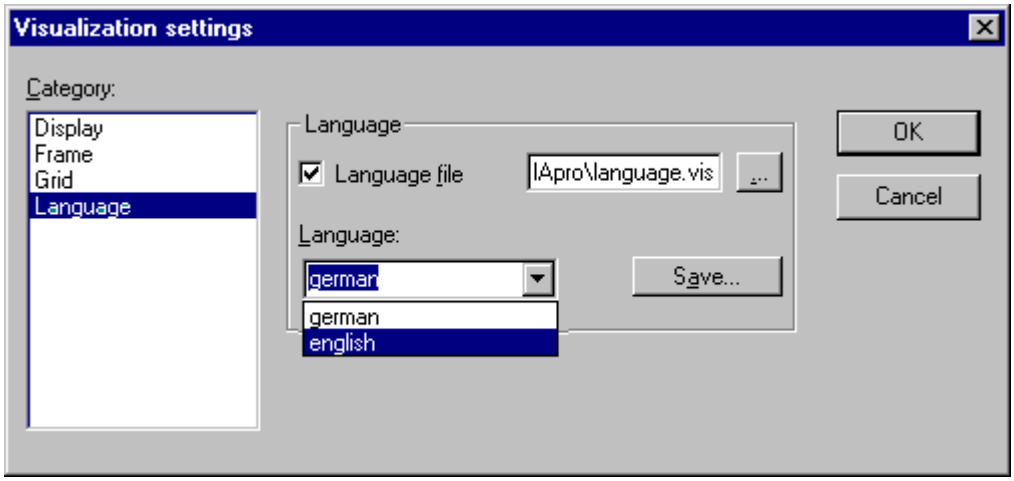


Image 7.20: Selection of a language file for a visualization

Creating a *.vis language file:

In order to set up a new *.vis language file for the visualization, proceed as follows:

Open likewise the **Settings** Visualization dialog, **Language** category.

Choose option language file. In the associate input field enter where you want to store the file. The extension is **.vis**. You also can use the dialog ‘Open file’ by pressing the button . If a language file with the extension .vis is already present, it will be offered to you here.

In the input field next to **Language** you fill in a keyword for the language which is currently used in the visualization, i.e.”german” (or “D”). then press the button **Save**. A file with the extension .vis will be created, which now can be edited by a normal text editor. For example you can open the file by NOTEPAD:

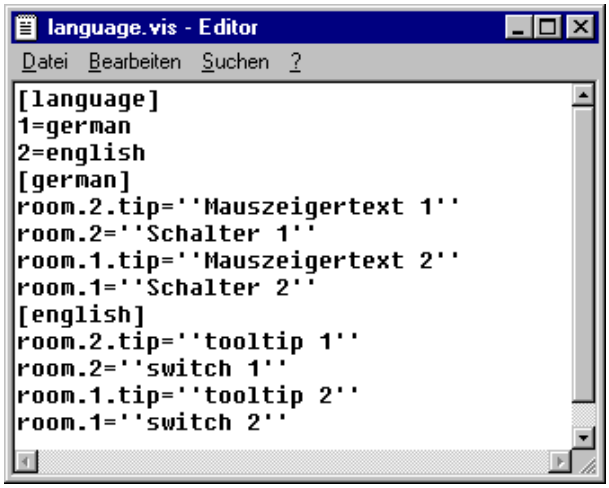


Image 7.21: Example of a language file for a visulisation (Category Language)

You get a list of the text variables for the language currently used in the visualization. It includes a reference to the title of this list, for example "1=german" as reference to the title [german]. You can extend the list by copying all lines, then replacing the German by English text and setting a new title [english]. Beyond the line 1=german you accordingly have to add 2=english.

To view the visualization in one of the prepared languages, open the dialog **Language** again. In the option field beyond **Language** now you can choose between *german* and *english* (for the example described above).

Note: The text display does not change before switching to Online Mode !

Calling up language-dependent Online Help via a visualization element:

The calling of a different Help file with a visualization element can be tied in with the language currently entered for the visualization. For this purpose, the command INTERN HELP must be entered for this element in the 'Configure element' dialog at the location 'Execute program', and a [Visu-Helpfiles] section must be present in the **CoDeSys** .ini file. Below this, the corresponding help files must be assigned to the languages available for selection in the visualization: e.g.:

```
[Visu-Helpfiles]
German=C:\PROGRAMME\HELP_D.HLP
English=C:\PROGRAMME\HELP_E.HLP
```

‘Extras‘ ‘Select Background Bitmap‘

Use this command to open the dialog box for selecting files. Select a file with the extension "*.bmp". The selected bitmap will then appear as the background in your visualization.

The bitmap can be removed with the command 'Extras' 'Clear Background Bitmap'.

‘Extras‘ ‘Clear Background Bitmap‘

Use this command to remove the bitmap as the background for the current visualization.

You can use the command 'Extras' 'Select Background Bitmap' to select a bitmap for the current visualization.

Keyboard usage‘

The use of hotkeys ‘can optimize the pure keyboard operation of a visualization.

In the configuration of a visualization object you can define hotkeys which will cause actions like visualization elements do. For example you could define that – if visualization 'xy' is active – in online mode the hotkey <Strg><F2> will stop the program, which also will happen as soon as element 'z' of visu 'xy' gets an input (by mouseclick or via touch screen).

Anyway per default the keys <Tabulator> <Space> <Enter> will work in that way that in online mode each element of a visualization can be selected and activated.

The dialog 'Keyboard usage: set possible keystrokes' can be called in the menu 'Extras' or in the context menu:

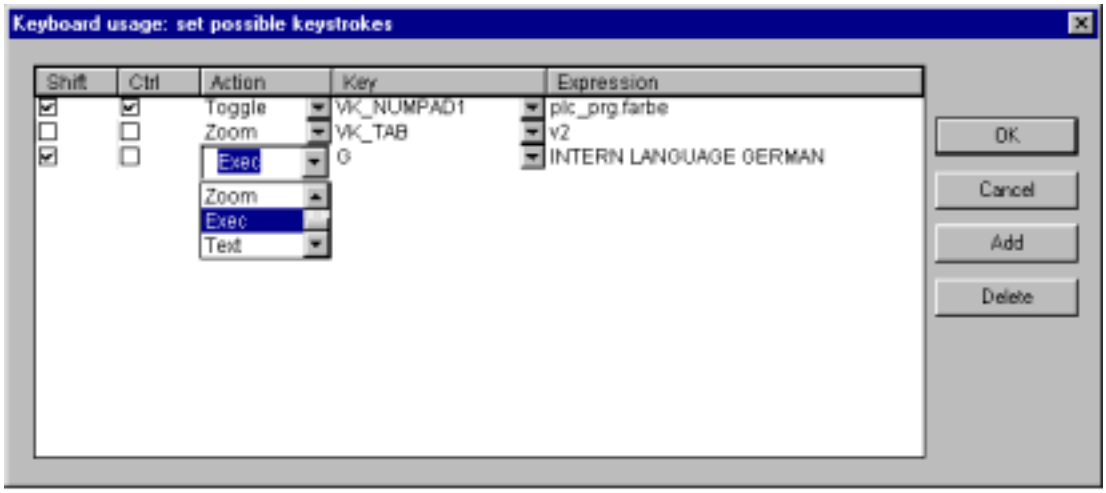


Image 7.22: Dialog 'Keyboard usage: set possible keystrokes'

In column Key a selection list offers the following keys to which an action can get assigned:

VK_TAB	Tab-Key
VK_RETURN	Enter-Key
VK_SPACE	Space-Key
VK_ESCAPE	Esc-Key
VK_INSERT	Insert-Key
VK_DELETE	Delete-Key
VK_HOME	Pos1-Key
VK_END	Ende-Key
VK_PRIOR	Bild (↑)-Key
VK_NEXT	Bild (↓)-Key
VK_LEFT	Arrow-Key (←)
VK_RIGHT	Arrow-Key (→)
VK_UP	Arrow-Key (↑)
VK_DOWN	Arrow-Key (↓)
VK_F1-VK_F12	Function keys F1 to F12
0-9	Keys 0 to 9
A-Z	Keys A to Z
VK_NUMPAD0 - VK_NUMPAD9	Keys 0 to 9 of the numeric keypad
VK_MULTIPLY	Key* of the numeric keypad
VK_ADD	Key+ of the numeric keypad
VK_SUBTRACT	Key- of the numeric keypad
VK_DIVIDE	Key÷ of the numeric keypad

In the columns **Shift** and **Ctrl** you can add the <Shift>- and/or the <Ctrl>-key to the already chosen key, so that a key combination will result.

In column **Action** you define what should happen as soon as the key (combination) will be pressed. Select the desired action from the list and insert an appropriate expression. See in the following the available actions and valid expressions, corresponding to those which can be set in the configuration dialog of category 'Input':

Action	Meaning	Expression
Toggle	Toggle variable	Variable, e.g."plc_prg.tvar"
Tap true	Tap variable (set to TRUE)	Program variable, e.g. "plc_prg.svar"
Tap false	Tap variable (set to FALSE)	Program variable, e.g. "plc_prg.xvar"
Zoom	Zoom to Vis.	Name of the visualization object to which you want to jump, e.g. "Visu1"
Exec	Execute program	Name of the executable file, e.g. "notepad C:\help.txt" (Notepad will start and open the file help.txt)
Text	Text input of variable 'Text display'	Number of the element for which the text input is to be configured, e.g. "#2" (Display of element numbers can be switched on in 'Extras' 'Settings'; also see 'Elementlist...')

In column **Expresion** you must enter – depending on the type of action – either a variable name, a INTERN command, a visualization name of a text string, exactly like you would do in the configuration dialog of category 'Input' for the corresponding visualization element.

Use button **Add** to add another empty line at the end of the table. Use the **Delete** button to remove the line where the cursor is positioned currently. **OK** resp. **Cancel** will save resp. not save the done settings and close the dialog.

The keyboard usage can be configured separately for each visualization object. Thus the same key (combination) can start different actions in different visualization.

Example:

The following key configurations have been done for the visualizations VIS_1 and VIS_2:

VIS_1:

Shift	Ctrl	Action	Key	Expression
x		Toggle	A	PLC_PRG.automatic
	x	Zoom	Z	VIS_2

VIS_2:

Shift	Ctrl	Action	Key	Expression
		Exec	E	INTERN LANGUAGE DEUTSCH
	x	Zoom	Z	PLC_VISU

If you now go online and set the focus to VIS_1, then pressing <Shift><Ctrl><A> will cause that variable PLC_PRG.automatic will be toggled. <Ctrl><Z> will cause a jump from Visu1 to VIS_2.

If VIS_2 is the active window, pressing key <E> will cause that the language within the visualizatoin will switch to German. <Ctrl><Z> here will cause a jump to visualization PLC_VISU.

7.2.3 Visualization in Online Mode

Regard the following items concerning a visualization in online mode:

- Order of evaluation:
  - Dynamically defined – that means by normal project variables or by structure variables- element properties will overwrite the (fix) base settings defined by options in the configuration dialogs.
  - If an element property is defined by a project variable as well as by the component of a structure variable , then primarily the value of the project variable will be regarded.
  - A visualization can be configured in that way that in online mode it can be operated solely by inputs via keyboard. This is an important feature especially for using the visualization with CoDeSys HMI, as Target- or as Web-Visualization.
- The configuration settings for Display, Frame and Language (see Chapter 5.2.1 'Settings') can also be edited in online mode.
- As long as a visualization reference is not configured explicitely when it is inserted in another visualization, the particular elements of the reference in online mode will react on inputs like those of the original visualization ("mother" of the references).
- When you switch the language ('Extras' 'Settings') this will only effect the display in online mode.A visualization can be printed in online mode.

Operation over the keyboard - in online mode

In order to get independent from the mouse or a touch screen, it is useful to configure a visualization in a way that allows pure keyboard operation:

Per default the following key (combinations) will work in online mode anyway (no special configuration necessary):

Pressing the <Tabulator> key selects the first element in the element list for which an input is configured. Each subsequent pressing of the key moves one to the next element in the list. Pressing the key while keeping the <Shift> key depressed selects the previous element.

The arrow keys can be used to change from a selected element to a neighbouring one in any direction.

The <**Space bar**> is used to execute an activity on the selected visualization element. If the element is one which has a text output variable, a text input field will be opened which displays the text contents of the variable. Pressing the <**Enter**> key writes in this value.

Additional key (combinations) for the online operation can be defined in the configuration dialog '**Keyboard usage**'. There also the keys <Tab>, <Space> and <Enter> can get assigned another functions than the above described standards.

The individual elements of references behave in Online mode identically to the corresponding ones in the visualization that is referenced. They will therefore react the same way as individual elements to inputs and operation by mouse and keyboard; the display of tooltips in references is also element-dependent. When processing the element list, as for instance when jumping from one input element to the next using the tabulator, the processing of all individual elements of a reference proceeds from

the location of the reference in the element list before jumping to the next element of the list.

Note: Operation over the keyboard in online mode is of greatest significance for the operation version of CoDeSys!

'File‘ 'Print' in online mode

‘File‘ ‘Print is used to print out the contents of the visualization window in online mode. Visualizations which stretch over the border of the window can lead to inconsistencies particularly when there are moving elements in the visualization.

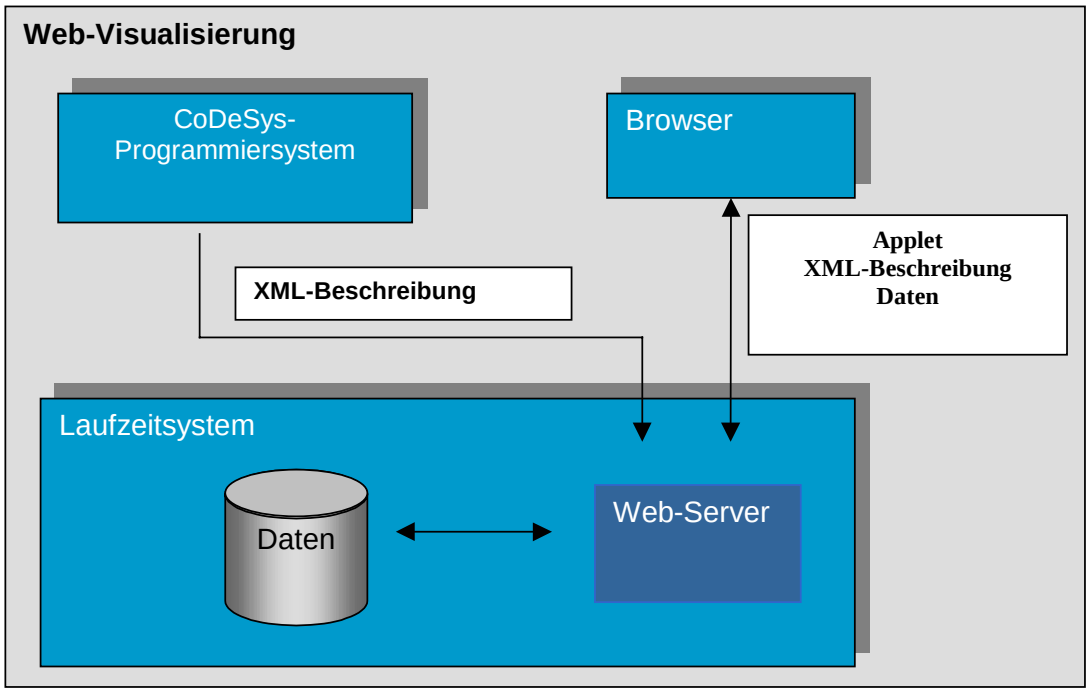
7.3 Visualizations in libraries

Visualizations can also be stored in libraries and thus be made available to projects as library POUs. They can be inserted as references like the visualizations directly present in the project, or they can be called up via the command „Zoom to vis.“ in the input configuration of another visualization.

Note: Visualizations used in a project must have unique names. It can be problematic if for instance a visualization from a library is called or referenced which has the same name as one present in the project. Because, in processing references or visualization calls in the program, first the visualizations in the project, and only thereafter the ones in the loaded libraries will be implemented.

7.4 Web-Visualization

CoDeSys can create XML descriptions of the visualization objects and download them to the PLC. There a Web-Server will provide the PLC data in XML format too and thus can create a currently updated visualization which can be opened in the Web Browser of any computer which is connected via Internet, independently from the target platform (e.g. useful for remote maintenance purposes).



Please regard:

Due to the low tranfer rates of the Internet only about 100 variables per visualization window can be monitored.

Preconditions:

- the target system must support the functionality; that means that in the target settings the option 'Web-Visualization' must be activated. If defined in the target file, this can be done by the user in the 'General' target settings.
- A **WebServer** must be started. (see below)
- In order to get the Web-Visualization displayed on a computer, a **Web-Browser** (IE-Explorer oder Netscape) is required.
- **Operating system**: Windows NT/2000, Windows CE, Linux, RTE

Preparing a visualization for Web:

- Create the visualization(s) for your PLC program as usual in the CoDeSys programming system. If you want a certain visualization to be called as starting object, then name it '**PLC_VISU**'. It will be loaded automatically as soon as the visualization is called via the Internet.
- For visualization objects, which should not be part of the web version of the visualization, deactivate option Web-Visualization' in the dialog **'Object' 'Properties'** in category 'Visualization'.
- Perform command 'Project' 'Clean all', then 'Project' 'Build'.
- Log in to the target system ('Project' **'Online' 'Login'**) and start the project on the PLC.

Starting the WebServer:

- An executable file of the WebServer must be available. It is called from the command line:

```
WebServer [Web Port-Number] [Target Port-Number]
```

```
[TargetIPAddress]
```

Web Port-Number:	Port number of the WebServer (default=80)
Target Port-Number:	Port number of the PLC (default=1200) to which the Webserver will connect.
TargetIPAddress:	IP-Address of the PLC (default=localhost) to which the Webserver will connect.

Example: `WebServer 8080 1200 192.168.100.11`

Calling a Web-Visualization via Internet:

- Insert the following address in the Browser:
  http://<IP-Adresse of the WebServers>:<Port of the WebServer>/webvisu.htm
- **WebVisu** is the default HTML-file. It contains a <applet> tag which will start the WebVisu-Applet.

7.5 Target-Visualization

CoDeSys can create ST code (Structured Text) for the visualization objects of a project. This code can be downloaded to the PLC together with the normal project code. Thus, if it is supported by the target system and if an appropriate monitor is available, then the visualization can be started directly on the PLC computer. It is not necessary any longer that the CoDeSys programming system is running in order to run the visualization. This means a considerable reduction of storage use.

Preconditions for the Target-Visualization:

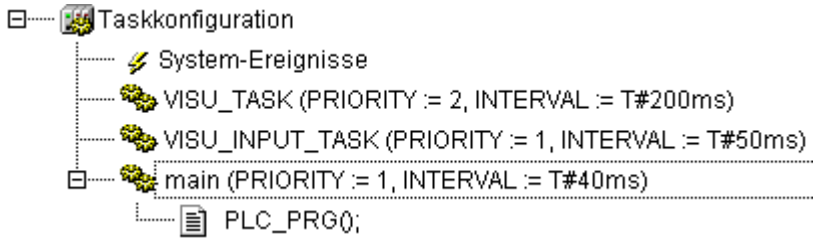
- The target system must support the functionality; that means that in the target settings the option 'Target-Visualization' must be activated. If defined in the target file, this can be done by the user in the 'General' target settings.
- The libraries **SysLibTargetVisuExtern.lib** (external) and **SysLibTargetVisuIntern.lib** (internal) are required for implementing the visualization functions in the runtime system. They will be linked to the project as soon as the option 'Target-Visualization' is activated in the target settings. In the runtime system the SysLibTargetVisuExtern.lib also must be implemented.
- Operating system of the PLC computer: Windows NT/2000, Windows CE
- The PLC computer needs devices for displaying and operating the visualization (monitor, keyboard etc.)

Creating a Target-Visualization

1. Create a visualization in CoDeSys. If there is a visualization object **PLC_VISU**, the Target-Visualization later will be started with this object. Otherwise it will start with that object which is the last one in the list of visualization objects in the Object Organizer.

   Please regard: For target computers running with Windows NT/2000, Windows CE the bitmaps of the visualization will be transferred as files.

2. For each visualization object, which should not be part of the Target-Visualization, deactivate the option 'Target-Visualization' in the **Object Properties dialog** ('Project' 'Object' 'Properties') in category 'Visualization'.
3. Make sure that the option 'Optimized jumpls' is deactivated in the target settings (Target Platform).
4. A project which is intended to be used with the Target-Visualization, must be controlled by a task configuration. If the option 'Target-Visualization' is activated in the target settings, then automatically two tasks are created: VISU_INPUT_TASK will do the processing of the user inputs, VISU_TASK will do the redrawing (update) of the visualization elements. See the default settings of the tasks in the example which is shown at the end of the paragraph. Of course the parameters can be modified. But: VISU_INPUT_TASK always should be processed before VISU_TASK in order to guarantee a useful interaction of user inputs and update of the visualization. That task, which calls the main program (e.g. PLC_PRG), should at least be processed as often as VISU_INPUT_TASK, ideally even with a higher priority, but it also could be added directly to VISU_INPUT_TASK


Taskkonfiguration
System-Ereignisse
VISU_TASK (PRIORITY := 2, INTERVAL := T#200ms)
VISU_INPUT_TASK (PRIORITY := 1, INTERVAL := T#50ms)
main (PRIORITY := 1, INTERVAL := T#40ms)
PLC_PRG();

5. **Load** the project to the PLC ('Online' 'Login').

Calling the Target-Visualization on the Target System:

Start the loaded project on the PLC. Hereupon the visualization will start with the PLC_VISU object resp. – if no PLC_VISU is available – with that visualization object which is the last one in the list of visualization objects in the Object Organizer. .

Please regard: Even if the option 'Target-Visualization' has been deactivated for each visualization object, nevertheless a visualization window will open as soon as the project is started on the PLC, if the visualization had been started before at least once with a visualization object.

8 DDE Interface

DDE Kommunikation mit
CoDeSys

CoDeSys has a DDE (dynamic data exchange) interface for reading data. **CoDeSys** uses this interface to provide other applications that also use a DDE Interface with the contents of control variables and IEC addresses

If the GatewayDDEServer is used, which works with symbols, CoDeSys is not needed to read variables values from the PLC and to transfer them to applications with an DDE interface.

Attention: Direct addresses cannot be read over the DDE server ! For this case you have to define variables in CoDeSys which are assigned to the desired addresses (AT).

Attention: The DDE interface has been tested with Word 97 and Excel 97 on Windows NT 4.0. If the DDE communication fails caused by a mismatch of other versions or additionally installed programs on a computer, 3S – Smart Software Solutions cannot take any responsibility.

8.1 DDE interface of the CoDeSys programming system

Activating the DDE Interface

The DDE interface becomes active as soon as the PLC (or the simulation) is logged in.

General Approach to Data

A DDE inquiry can be divided into three parts:
1. Name of the program (here: **CoDeSys**),
2. File name and
3. Variable name to be read.

Name of the program: **CoDeSys**

File name: complete project path (c:\example\example.pro).

Variable name: The name of a variable as it appears in the Watch and Receipt Manager .

Which variables can be read?

All addresses and variables are readable. Variables or addresses should be entered in the format used in the Watch and Receipt Manager

Examples:

```
%IX1.4.1          (* Reads the input 1.4.1*)
PLC_PRG.TEST      (* Reads the variable TEST from the POU PLC_PRG*)
.GlobVar1         (* Reads the global variable GlobVar1 *)
```

Linking variables using WORD

In order to get the current value of the variable TEST from the POU PLC_PRG through the DDE interface in Microsoft WORD, a field (e.g., the date) must be inserted in WORD ('Insert' "Field"). Now when you click on the field with the right mouse button and select the command "Toggle Field Codes" you can change the field function for the chosen text. In our example, this would look as follows:

```
{ DDEAUTO CODESYS"C:\CODESYS\PROJECT\IFMBSP.PRO"
  "PLC_PRG.TEST" }
```

Click on the field with the right mouse button again, then click on "Update Field" and the desired variable content appears in the text.

Linking variables using EXCEL

The following must be entered in Microsoft EXCEL before you can assign a variable to a cell.

```
=CODESYS|'C:\CODESYS\PROJECT\IFMBSP.PRO'!PLC_PRG.TEST'
```

When you click on 'Edit' then "Links", the result for this link will be:

```
Type: CODESYS
Source file: C:\CODESYS\PROJECT\IFMBSP.PRO
Element: PLC_PRG.TEST
```

Accessing variables with Intouch

Link with your project a DDE Access Name <AccessName> with the application name CODESYS and the DDE topic name C:\CODESYS\PROJECT\IFMBSP.PRO.

Now you can associate DDE type variables with the access name <AccessName>. Enter the name of the variable as the Item Name (e.g., PLC_PRG.TEST).

8.2 DDE communcation with the GatewayDDE Server

Handling of the GatewayDDE Server

The GatewayDDE Server can use the symbols which are created in CoDeSys for a project to communciate with other clients or the PLC. (see 'Project' 'Options' 'Symbolconfiguration'). It can serve the DDE interfaces of applications like e.g. Excel. This allows to transmit the variables values of the PLC to an applications, e.g. for the purpose of monitoring.

At start of the GatewayDDE Server a window opens, where the configuration of start and communication parameters can be done. A already existing configuration file can be called or the parameters can be set newly.

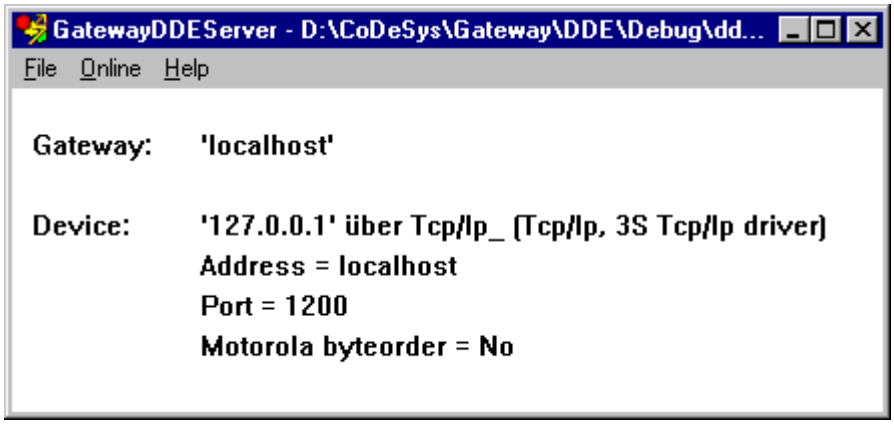


Image 8.1: :Starting Window of the GatewayDDE Server

Using the command **'File' 'Open'** you can call an already existing file which stores a set of configuration parameters. The standard dialog for selecting a file will open and available files with the extension ".cfg" will be offered. If a configuration file is selected, the configuration parameters and the defined target device are displayed

If the option **'File' 'Autoload'** is activated, the GatewayDDE Server automatically opens with that configuration, which was active before the last terminating of the server.

If the server is started without any predefined configuration and without the setting Autoload, then in the configuration window 'Gateway:' und 'Device:' are displayed. Then you have to set up a new configuration.

The command **'File' 'Settings'** opens the dialog '**Server settings**', in which the following parameters can be set:

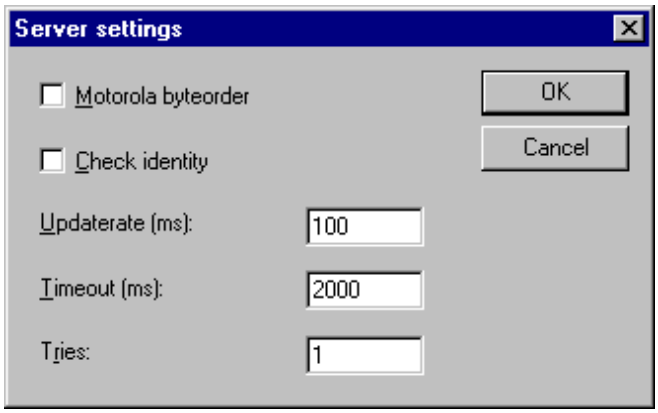


Image 8.2: Dialog for configuring the GatewayDDE Server

To set up the connection to the Gateway, the dialog **'Communication Parameters'** is opened by the command 'Online' 'Parameters'. It is the same dialog as you get in **CoDeSys** with the command **'Online' 'Kommunikationsparameter'**. The settings you do here must be the same as in the corresponding **CoDeSys** Project.

The actual configuraiton of the GatewayDDE Server can be stored in a file by the command **'File' 'Save'**. The standard dialog for saving a file will open, default for the extension of the file is *.cfg.

To get the gateway in active mode, login by the command **'Online' 'Login'**. (The gateway symbol in the status bar will get lightened then.) At login the desired connection will be built up and the available symbols can be accessed.. These must have been created before in the **CoDeSys** Project !

To log out use the command **'Online' Logout'**.

General Approach to Data

A DDE inquiry can be divided into three parts:

1. Name of the program (here: **CoDeSys**),
2. File name and
3. Variable name to be read.

Name of the program: **CoDeSys**

File name: complete project path (c:\example\example.pro).

Variable name: The name of a variable as it appears in the Watch and Receipt Manager .

Which variables can be read?

All addresses and variables are readable. Variables or addresses should be entered in the format used in the Watch and Receipt Manager

Examples:

```
%IX1.4.1        (* Reads the input 1.4.1*)
PLC_PRG.TEST    (* Reads the variable TEST from the POU PLC_PRG*)
.GlobVar1       (* Reads the global variable GlobVar1 *)
```

Linking variables using WORD

Start the GatewayDDEServer before activating the inquiry in WORD.

In order to get the current value of the variable TEST from the POU PLC_PRG through the DDE interface in Microsoft WORD, a field (e.g., the date) must be inserted in WORD ('Insert' "Field"). Now when you click on the field with the right mouse button and select the command "Toggle Field Codes" you can change the field function for the chosen text. In our example, this would look as follows:

```
{ DDEAUTO GATEWAYDDESERVER "BSP.PRO" "PLC_PRG.TEST" }
```

Click on the field with the right mouse button again, then click on "Update Field" and the desired variable content appears in the text.

Linking variables using EXCEL

Start the GatewayDDEServer before activating the inquiry in EXCEL.

The following must be entered in Microsoft EXCEL before you can assign a variable to a cell.

```
=GATEWAYDDESERVER|<Dateiname>!<Variablenname>
Beispiel:
=GATEWAYDDESERVER|'bsp.pro'!'PLC_PRG.TEST'
```

When you click on 'Edit' then "Links", the result for this link will be:

```
Type: CODESYS
```

Source file: C:\CODESYS\PROJECT\IFMBSP.PRO
Element: PLC_PRG.TEST
{ DDEAUTO GATEWAYDDESERVER "BSP.PRO" "PLC_PRG.TEST" }

Command line options for the GatewayDDEServer

If the GatewayDDE Server is started by a command line, the following options can be attached:

/n	The info dialog does not appear automatically at starting		
/s	Display of the dialog window	/s=h /s=i /s=m /s=n	No minimized (icon) maximized normal
/c	Configuration file to be load automatically l	/c=<config-file>	
/o	Go online with the selected configuration (autoload or defined by "/c=")		

Example:

Command line:

GATEWAYDDE /s=i /c="D:\DDE\conf_1.cfg"

The GatewayDDE Server will be started, the dialog window will appear as an icon, the configuration which is stored in the file conf_1.cfg will be loaded.

9 The CoDeSys ENI

What is ENI

The ENI ('Engineering Interface') allows to connect the CoDeSys programming system to an **external data base**. There the data which are needed during creation of an automation project can be stored. The usage of an external data base guarantees the consistency of the data, which then can be shared by several users, projects and programs. Also it extends the CoDeSys functionality by making possible the following items:

- **Revision control** for CoDeSys projects and associated resources (shared objects): If a object has been checked out from the data base, modified and checked in again, then in the data base a new version of the object will be created, but the older versions will be kept also and can be called again on demand. For each object and for a whole project the version history will be logged. Two versions can be checked for differences.
- **Multi-User Operation**: The latest version of a sample of objects, e.g. of POUs of a project, can be made accessible for a group of users. Then the objects which are currently checked out by one of the users will be marked as "in the works" and not editable by the other users. Thus several users can work in parallel on the same project without risking to overwrite versions mutually.
- **Access by external tools**: Besides the CoDeSys programming system also other tools, which have an ENI too, can access the common data base. These might be e.g. external visualizations, ECAD systems etc., which need the data created in CoDeSys or which also produce data which are needed by other programs.

The ENI is composed of a **client** and a **server** part. So it is possible to hold the data base on a remote computer, which is required for multi-user operation. The CoDeSys programming system is a client of the independent **ENI Server Process** as well as another application, which needs access to the data base (Please see the separate documentation on the *ENI Server*).

Currently the ENI supports the data base systems 'Visual SourceSafe 5.0', 'Visual SourceSafe 6.0', 'MKS Source Intergrity' and a local file system. Objects can be stored there in different 'folders' (**data base categories** with different access properties, see Chapter 7.3). The objects can be checked out for editing and thereby will get locked for other users. The latest version of an object can be called from the data base. Furtheron in parallel you can store any objects just locally in the project as usual for projects without source control.

9.1 Preconditions for Working with an ENI project data base

If you want to use the ENI in the CoDeSys programming system in order to manage the project objects in an external data base, the below mentioned preconditions must be fulfilled:

Please regard: For a guide concerning installation and usage of the *ENI Server*

provided by 3S – Smart Software Solutions GmbH please see the separate server documentation resp. online help. There you will also find a quickstart guide. Also consider the possibility of using the *ENI Explorer* which allows to perform data base actions independently from the currently used data base system.

- the communication between CoDeSys and the ENI Server requires **TCP/IP**, because the *ENI Server* uses the HTTP-Protokoll.
- an *ENI Server* (*ENI Server Suite*)) must be installed and started locally or on a remote computer. A license is required to run it with the standard database drivers which are installed with the server. Just the driver for a local file system can be used with a non-licensed ENI Server version. (see Chapter 8, 'License Management in CoDeSys').
- in the *ENI Server* administration tool (**ENI Admin**) the following must be configured:
  - the user must be registered and have access rights (User Management)
  - the access rights concerning the data base folders must be set correctly (Access Rights)

  – Recommendation: the administrator passwort for the access to ENIaAdmin and ENI Control should be defined immediately after the installation
- in the *ENI Server* service control tool (**ENI Control**) the connection to the desired data base must be configured correctly (Data base). You will automatically be asked to do this during installation, but you can modify the settings later in *ENI Control*.
- a **project data base** for which an ENI-supported driver is available, must be installed. It is reasonable to do this on the same computer, where the *ENI Server* is running. Alternatively a local file system can be used, for which a driver will be provided by default.
- in the **data base administration** possibly the user (Client) as well as the ENI Server must be registered as valid users with access rights. Anyway this is required for the 'Visual SourceSafe', if you use another data base system, please see the corresponding documentation for information on the user configuration.
- for the current CoDeSys project the **ENI interface must be activated** (to be done in the CoDeSys dialog 'Project' 'Options' 'Project data base').
- for the current CoDeSys project the **connection to the data base** must be configured correctly; this is to be done in the CoDeSys dialogs 'Project' 'Options' 'Project data base'.
- in the current project the user must **log in to the *ENI Server*** with user name and password; this is to be done in the Login dialog, which can be opened explicitely by the command 'Project' 'Data Base Link' 'Login' resp. which will be opened automatically in case you try to access the data base without having logged in before.

(See also a quickstart guide on this in the document 'ENI Server – Overview and Quickstart')

9.2 Working with the ENI project data base in CoDeSys

The data base commands (Get Latest Version,Check Out, Check In, Version History, Label Version etc.) which are used for managing the project objects in the ENI

project data base, will be available in the current CoDeSys project as soon as the connection to the data base has been activated and configured correctly. See for this Chapter 7.1, Preconditions for Working with an ENI project data base. The commands then are disposable in the submenu 'Data Base Link' of the context menu or of the 'Project' menu and refer to the object which is currently marked in the Object Organizer.

The current assignment of an object to a data base category is shown in the Object Properties and can be modified there.

The properties of the data base categories (communication parameters, access rights, check in/check out behaviour) can be modified in the option dialogs of the project data base ('Project' 'Options' 'Project Source Control').

9.3 Object categories concerning the project data base

There are four categories of objects of a CoDeSys project concerning the project source control:

- The ENI distinguishes three categories ("ENI object categories") of objects which are managed in the project data base: Project objects, Shared objects, Compile files.
- If an object should not be stored in the data base, it will be assigned to category 'Local', which means that it will be handled as it is known for projects without any source control.

Thus in the programming system a CoDeSys object can be assigned to one of the categories 'Project objects', 'Shared objects' or 'Local'; the 'Compile files' do not yet exist as objects within the project. Assigning an object to one of the categories is done automatically when the object is created, because this is defined in the project options dialog 'Project source control' (see Chapter 2.2, but it can be modified anytime in the 'Object Properties' dialog.

Each ENI object category will be configured separately in the settings for the 'Project source control' which are part of the project options ('Project' 'Options'). That means that each category gets defined own parameters for the communication with the data base (directory, port, access right, user access data etc.) and concerning the behaviour at calling the latest version, checking out and checking in. These settings then will be valid for all objects belonging to the category. As a result you will have to log in to each data base category separately; this will be done via the 'Login' dialog.

It is advisable to create a separate folder for each object category in the data base, but it is also possible to store all objects in the same folder. (The 'category' is a property of an object, not of a folder.)

See in the following the three ENI object categories:

Project Objects: Objects which contain project specific source information, e.g. POUs which are shared in a multi-user operation. The command 'Get all latest versions' automatically will call all objects of this category from the data base to the local project; even those, which have not been there so far.

Shared Objects:	Objects which are not project specific, e.g. POU libraries which normally are used in several projects. Attention: The command 'Get all Latest Versions' only will copy those objects of this category from the project folder to the local project, which are already part of the project !
Compile files:	Compile information (e.g. symbol files) which is created by CoDeSys for the current project and which may be needed by other programs too. Example: An external visualization may need not only the project variables, but also the assigned addresses. The latter will not be known until the project is compiled.

Alternatively any objects of the CoDeSys project can be excluded from the project source control and can be assigned to category 'Local', which means that they are just stored with the project as usual for projects without any source control.

10 License Management in CoDeSys

The 3S Licensing Manager") is available to handle the licenses for 3S modules, as well as licenses for modules for which an appropriate license information file is provided, on your computer. In CoDeSys you can create a project and provide it as a licensed library. The Licensing Manager will be installed automatically with any 3S module, which requires a license.

10.1 Creating a licensed library in CoDeSys

As is known a CoDeSys project") ")can be saved as a library. If you want to create a licensed library you have to add the appropriate **license information**. For this perform the command **'File' 'Save as...'**, choose data type 'Internal Library' or 'External Library' and press button **Edit license info....**

In the dialog **Edit Licensing Information** enter the information described below. The license information will be added to the Project Info. When later on the library will be included in a CoDeSys project, the license information can be checked up in the object properties dialog of the library in the library manager.

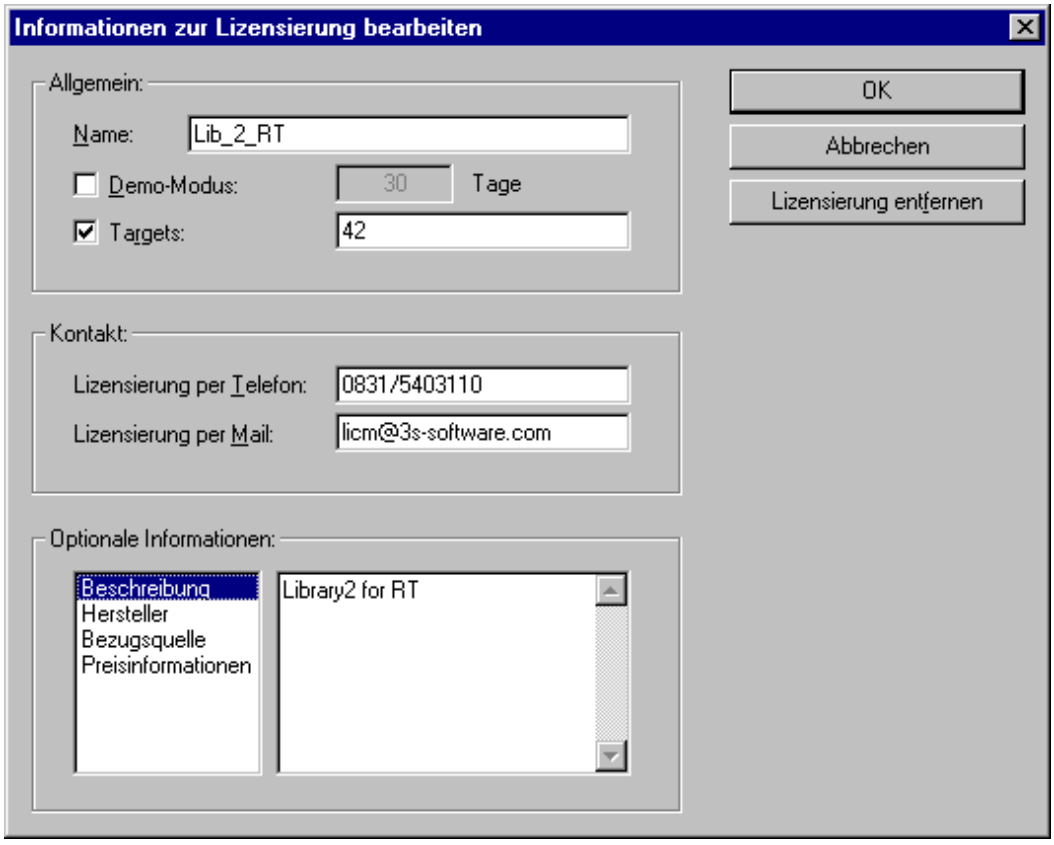


Image 10.1: Dialog: Edit Licensing Information

Common:

Name: Enter a name for the library module which is used to represent it in the *3S Licensing Manager*. This input is mandatory.

Demo mode: Activate this option if the module should be usable in demo mode, that means without any license ID. Enter the number of **days** after which the "demo license" should expire. The number of days will be automatically rounded up to the next number which is divisible by ten (10, 20, 30 ...). If no number is entered here, the module will be usable without time limitation !

Targets: Enter here the target-ID(s) of the target system(s) for which the license should be valid. Multiple inputs must be separated by a comma or a semicolon.

Contact:

Licensing via phone: / **Licensing per via mail:** Insert here the phone number resp. email address of the license provider. These inputs are mandatory.

Optional information:

In the right window you can enter a text referring to the item currently marked in the left window:

Description, Manufacturer, Vendor, Pricing information

Please note:

- It is reasonable to protect a library, which has been provided with licensing information, by a password (see Chapter 2.3, Passwords for User Groups). If you are going to save the project without password you will be pointed to that by a message box.
- The licensing information of a 3S library is stored internally with the library and will be registered on the computer automatically as soon as the library is included in a project. But the license information of modules which are not provided by 3S must be provided in a separate **description file** in compatible XML format, which can be read by the 3S Licensing Manager. For this also see the separate documentation on the ***3S License Manager*.**

11 APPENDIX

Appendix A: Use of Keyboard

If you would like to run CoDeSys using only the keyboard, you will find it necessary to use a few commands that are not found in the menu.

- The function key <F6> allows you to toggle back and forth within the open POU between the Declaration and the Instruction parts.
- <Alt>+<F6> allows you to move from an open object to the Object Organizer and from there to the Message window if it is open. If a Search box is open, <Alt>+<F6> allows you to switch from Object Organizer to the Search box and from the there back to the object.
- Press <Tab> to move through the input fields and buttons in the dialog boxes.
- The arrow keys allow you to move through the register cards and objects within the Object Organizer and Library Manager.

All other actions can be performed using the menu commands or with the shortcuts listed after the menu commands. **<Shift>+<F10>** opens the context menu which contains the commands most frequently used for the selected object or for the active editor.

Key Combinations

The following is an overview of all key combinations and function keys:

General Functions	
Move between the declaration part and the instruction part of a POU	<F6>
Move between the Object Organizer, the object and the message window	<Alt>+<F6>
Context Menu	<Shift>+<F10>
Shortcut mode for declarations	<Ctrl>+<Enter>
Move from a message in the Message window back to the original position in the editor	<Enter>
Open and close multi-layered variables	<Enter>
Open and close folders	<Enter>
Switch register cards in the Object Organizer and the Library Manager	<Arrow keys>
Move to the next field within a dialog box	<Tab>
Context sensitive Help	<F1>

General Commands	
‘File’ ‘Save’	<Ctrl>+<S>
‘File’ ‘Print’	<Ctrl>+<P>
‘File’ ‘Exit’	<Alt>+<F4>
‘Project’ ‘Check’	<Strg>+<F11>
‘Project’ ‘Build’	<Umschalt>+<F11>
‘Project’ ‘Rebuild all’	<F11>
‘Project’ ‘Delete Object’	<Del>
‘Project’ ‘Add Object’	<Ins>
‘Project’ ‘Rename Object’	<Spacebar>
‘Project’ ‘Open Object’	<Enter>
‘Edit’ ‘Undo’	<Ctrl>+<Z>
‘Edit’ ‘Redo’	<Ctrl>+<Y>
‘Edit’ ‘Cut’	<Ctrl>+<X> or <Shift>+<Del>
‘Edit’ ‘Copy’	<Ctrl>+<C>
‘Edit’ ‘Paste’	<Ctrl>+<V>
‘Edit’ ‘Delete’	<Del>
‘Edit’ ‘Find next’	<F3>
‘Edit’ ‘Input Assistant’	<F2>
‘Edit’ ‘Next Error’	<F4>
‘Edit’ ‘Previous Error’	<Shift>+<F4>
‘Online’ ’Log-in’	<Alt><F8>
‘Online’ ’Logout’	<Ctrl>+<F8>
‘Online’ ‘Run’	<F5>
‘Online’ ‘Toggle Breakpoint’	<F9>
‘Online’ ‘Step over’	<F10>
‘Online’ ‘Step in’	<F8>
‘Online’ ‘Single Cycle’	<Ctrl>+<F5>
‘Online’ ‘Write Values’	<Ctrl>+<F7>
‘Online’ ‘Force Values’	<F7>
‘Online’ ‘Release Force’	<Shift>+<F7>
'Online’ ‘'Write/Force dialog'	<Shift>+<F7>
‘Window’ ‘Messages’	<Shift>+<Esc>

FBD Editor Commands	
‘Insert’ ‘Network (after)’	<Shift>+<T>
‘Insert’ ‘Assignment’	<Ctrl>+<A>
‘Insert’ ‘Jump’	<Ctrl>+<L>
‘Insert’ ‘Return’	<Ctrl>+<R>
‘Insert’ ‘Operator’	<Ctrl>+<O>
‘Insert’ ‘Function’	<Ctrl>+<F>
‘Insert’ ‘Function Block’	<Ctrl>+<B>
‘Insert’ ‘Input’	<Ctrl>+<U>
‘Extras’ ‘Negate’	<Ctrl>+<N>
‘Extras’ ‘Zoom’	<Alt>+<Enter>
CFC Editor Commands	
‘Insert’ ‘POU’	<Ctrl>+<B>
'Insert‘ 'Input‘	<Ctrl>+<E>
‘Insert’ ‘Output’	<Ctrl>+<A>
‘Insert’ ‘Jump‘	<Ctrl>+<G>
‘Insert’ ‘Label‘	<Ctrl>+<L>
‘Insert’ ‘Return’	<Ctrl>+<R>
‘Insert’ ‘Comment’	<Ctrl>+<K>
‘Insert’ ‘POU input’	<Ctrl>+<U>
'Extras' 'Negate'	<Ctrl>+<N>
‘Extras’ ‘Set/Reset‘	<Ctrl>+<T>
‘Extras’ ‘Connection’	<Ctrl>+<M>
‘Extras’ ‘EN/ENO‘	<Ctrl>+<O>
'Extras' 'Zoom'	<Alt>+<Enter>
LD Editor Commands	
‘Insert’ ‘Network (after)’	<Shift>+<T>
‘Insert’ ‘Contact’	<Ctrl>+<O>
‘Insert’ ‘Parallel Contact’	<Ctrl>+<R>
‘Insert’ ‘Function Block’	<Ctrl>+<B>
‘Insert’ ‘Coil’	<Ctrl>+<L>
‘Extras’ ’Paste below’	<Ctrl>+<U>

‘Extras’ ‘Negate’	<Ctrl>+<N>
'Extras' 'Zoom'	<Alt>+<Enter>
SFC Editor Commands	
‘Insert’ ‘Step-Transition (before)’	<Ctrl>+<T>
‘Insert’ ‘Step-Transition (after)’	<Ctrl>+<E>
‘Insert’ ‘Alternative Branch (right)’	<Ctrl>+<A>
‘Insert’ ‘Parallel Branch (right)’	<Ctrl>+<L>
‘Insert’ ‘Jump’	<Ctrl>+<U>
‘Extras’ ‘Zoom Action/Transition’	<Alt>+<Enter>
Move back to the editor from the SFC Overview	<Enter>
Work with the PLC Configuration	
Open and close organization elements	<Enter>
Place an edit control box around the name	<Spacebar>
‘Extras’ ‘Edit Entry’	<Enter>
Work with the Task Configuration	
Place an edit control box around the task name or the- or program name	<Spacebar>
Work with the OD editor	
Focus each time on other tab	<F6>
Jump to the next editing field	Arrow keys
Append a new line with default values, if the cursor is in a cell of the lowest line	<Enter>
Place an edit control box around the task or program name	<Spacebar>

Appendix B: Data Types

Standard Data types

Data types

You can use standard data types and user-defined data types when programming. Each identifier is assigned to a data type which dictates how much memory space will be reserved and what type of values it stores.

BOOL

BOOL type variables may be given the values TRUE and FALSE. 8 bits of memory space will be reserved.

(see also: BOOL constants)

Integer Data Types

BYTE, **WORD**, **DWORD**, **SINT**, **USINT**, **INT**, **UINT**, **DINT**, and **UDINT** are all integer data types

Each of the different number types covers a different range of values. The following range limitations apply to the integer data types:

Type	Lower limit	Upper limit	Memory space
BYTE	0	255	8 Bit
WORD	0	65535	16 Bit
DWORD	0	4294967295	32 Bit
SINT:	-128	127	8 Bit
USINT:	0	255	8 Bit
INT:	-32768	32767	16 Bit
UINT:	0	65535	16 Bit
DINT:	-2147483648	2147483647	32 Bit
UDINT:	0	4294967295	32 Bit

As a result when larger types are converted to smaller types, information may be lost.

see also Appendix Appendix F: : Number Constants

REAL / LREAL

REAL and **LREAL** are so-called floating-point types. They are required to represent rational numbers. 32 bits of memory space is reserved for REAL and 64 bits for LREAL.

see also Appendix Appendix F: : REAL-/LREAL constants

STRING

A **STRING** type variable can contain any string of characters. The size entry in the declaration determines how much memory space should be reserved for the variable. It refers to the number of characters in the string and can be placed in parentheses or square brackets. If no size specification is given, the default size of 80 characters will be used.

Example of a String Declaration with 35 characters:

```
str:STRING(35):='This is a String';
```

see also Appendix Appendix F: : STRING constants

Time Data Types

The data types **TIME**, **TIME_OF_DAY** (abb. **TOD**), **DATE** and **DATE_AND_TIME** (abb. **DT**) are handled internally like DWORD.

Time is given in milliseconds in TIME and TOD, time in TOD begins at 12:00 A.M.

Time is given in seconds in DATE and DT beginning with January 1, 1970 at 12:00 A.M.

The time data formats used to assign values are described in the chapter 'Operands in CoDeSys, Constants'.

see also Appendix Appendix F: : TIME-/DATE-/TIME_OF_DAY/DATE_AND_TIME constants

Defined Data Types

ARRAY

One-, two-, and three-dimensional fields (arrays) are supported as elementary data types. Arrays can be defined both in the declaration part of a POU and in the global variable lists.

Syntax:

```
<Field_Name>:ARRAY [<ll1>..<ul1>,<ll2>..<ul2>] OF <elem. Type>.
```

ll1, ll2, ll3identify the lower limit of the field range; ul1, ul2 and ul3 identify the upper limit. The range values must be integers.

Example:

```
Card_game: ARRAY [1..13, 1..4] OF INT;
```

Initializing Arrays:

Example for complete initialization of an array:

```
arr1 : ARRAY [1..5] OF INT := 1,2,3,4,5;
arr2 : ARRAY [1..2,3..4] OF INT := 1,3(7);     (* short for 1,7,7,7 *)
arr3 : ARRAY [1..2,2..3,3..4] OF INT := 2(0),4(4),2,3;
                                        (* short for 0,0,4,4,4,4,2,3 *)
```

Example of the initialization of an array of a structure:

```
TYPE STRUCT1
STRUCT
```

```
p1:int;
p2:int;
p3:dword;
END_STRUCT

ARRAY[1..3] OF STRUCT1:= (p1:=1,p2:=10,p3:=4723),(p1:=2,p2:=0,p3:=299),
   (p1:=14,p2:=5,p3:=112);
```

Example of the partial initialization of an Array:

```
arr1 : ARRAY [1..10] OF INT := 1,2;
```

Elements to which no value is pre-assigned are initialized with the default initial value of the basic type. In the example above, the elements anarray[6] to anarray[10] are therefore initialized with 0.

Array components are accessed in a two-dimensional array using the following syntax:

```
<Field_Name>[Index1,Index2]
```

Example:

```
Card_game [9,2]
```

Note: If you define a function in your project with the name **CheckBounds**, you can automatically check for out-of-range errors in arrays !

The name of the function is fixed and can only have this designation.

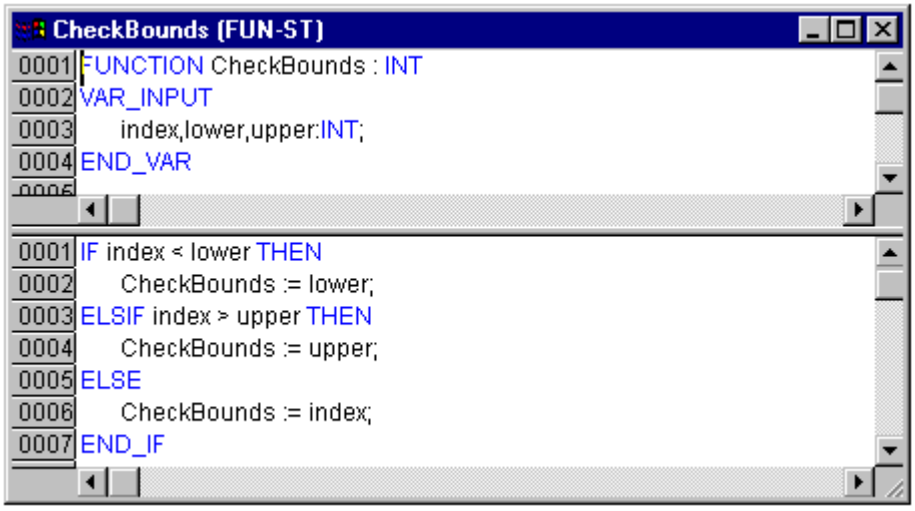


Image Appendix B: .2: Example for an implementation of the CheckBounds function

The following sample program for testing the CheckBounds function exceeds the bounds of a defined array. The CheckBounds function allows the value TRUE to be assigned, not to location A[10], but to the still valid range boundary A[7] above it. With the CheckBounds function, references outside of array boundaries can thus be corrected.

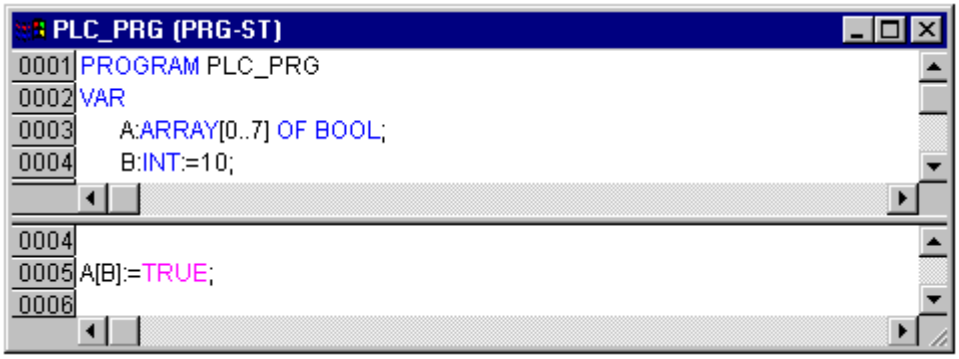


Image Appendix B: .3: Test program for the function CheckBounds

Pointer

Variable or function block addresses are saved in pointers while a program is running.

Pointer declarations have the following syntax:
<Identifier>: **POINTER TO** <Datatype/Functionblock>;

A pointer can point to any data type or function block even to user-defined types.

The function of the Address Operator ADR is to assign the address of a variable or function block to the pointer.

A pointer can be dereferenced by adding the content operator "^" after the pointer identifier.

Example:

```
pt:POINTER TO INT;
var_int1:INT := 5;
var_int2:INT;
pt := ADR(var_int1);
var_int2:= pt^; (* var_int2 is now 5 *)
```

Enumeration

Enumeration is a user-defined data type that is made up of a number of string constants. These constants are referred to as enumeration values.

Enumeration values are recognized in all areas of the project even if they were declared within a POU. It is best to create your enumerations as objects in the Object Organizer under the register card **Data types**. They begin with the keyword TYPE and end with END_TYPE.

Syntax:

```
TYPE <Identifier>:(<Enum_0> ,<Enum_1>, ...,<Enum_n>);
END_TYPE
```

A variable of the type <Identifier> can take on one of the enumeration values and will be initialized with the first one. These values are compatible with whole numbers which means that you can perform operations with them just as you would with INT. You can assign a number x to the variable. If the enumeration values are not initialized, counting will begin with 0. When initializing, make certain the initial values are increasing. The validity of the number will be reviewed at the time it is run.

Example:

```
TYPE TRAFFIC_SIGNAL: (Red, Yellow, Green:=10); (*The initial value for
    each of the colors is red 0, yellow 1, green 10 *)
END_TYPE
TRAFFIC_SIGNAL1 : TRAFFIC_SIGNAL;
TRAFFIC_SIGNAL1:=0; (* The value of the traffic signal is red*)
FOR i:= Red TO Green DO
    i := i + 1;
END_FOR;
```

The same enumeration value could not be used twice.

Example:

```
TRAFFIC_SIGNAL: (red, yellow, green);
COLOR: (blue, white, red);
```

Error: red may not be used for both TRAFFIC_SIGNAL and COLOR.

Structures

Structures are created as objects in the Object Organizer under the register card **Data types**. They begin with the keywords TYPE and STRUCT and end with END_STRUCT and END_TYPE.

The syntax for structure declarations is as follows:

```
TYPE <Structurename>:
STRUCT
    <Declaration of Variables 1>
    .
    .
    <Declaration of Variables n>
END_STRUCT
END_TYPE
```

<Structurename> is a type that is recognized throughout the project and can be used like a standard data type.

Interlocking structures are allowed. The only restriction is that variables may not be placed at addresses (the AT declaration is not allowed!).

Example for a structure definition named Polygonline:

```
TYPE Polygonline:
STRUCT
    Start:ARRAY [1..2] OF INT;
    Point1:ARRAY [1..2] OF INT;
    Point2:ARRAY [1..2] OF INT;
    Point3:ARRAY [1..2] OF INT;
    Point4:ARRAY [1..2] OF INT;
    End:ARRAY [1..2] OF INT;
END_STRUCT
END_TYPE
```

Example for the initialization of a structure:

```
Poly_1:polygonline := ( Start:=3,3, Point1 =5,2, Point2:=7,3, Point3:=8,5,
    Point4:=5,7, End := 3,5);
```

Initializations with variables are not possible. See an example of the initialization of an array of a structure under 'Arrays'.

You can gain access to structure components using the following syntax:
<Structure_Name>.<Componentname>

For example, if you have a structure named "Week" that contains a component named "Monday", you can get to it by doing the following:
Week.Monday

References

You can use the user-defined reference data type to create an alternative name for a variable, constant or function block.

Create your references as objects in the Object Organizer under the register card **Data types**. They begin with the keyword TYPE and end with END_TYPE.

Syntax:
TYPE <Identifier>: <Assignment term>;
END_TYPE

Example:
TYPE message:STRING[50];
END_TYPE;

Subrange types

A subrange type is a type whose range of values is only a subset of that of the basic type. The declaration can be carried out in the data types register, but a variable can also be directly declared with a subrange type:

Syntax for the declaration in the 'Data types' register:

TYPE <Name> : <Inttype> (<ug>..<og>) END_TYPE;

<Name> must be a valid IEC identifier,

<Inttype> is one of the data types SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD (LINT, ULINT, LWORD).

<ug> Is a constant which must be compatible with the basic type and which sets the lower boundary of the range types. The lower boundary itself is included in this range.

<og> Is a constant that must be compatible with the basic type, and sets the upper boundary of the range types. The upper boundary itself is included in this basic type.

Example:
```
TYPE
   SubInt : INT (-4095..4095);
END_TYPE
```

Direct declaration of a variable with a subrange type:
```
VAR
   i : INT (-4095..4095);
   ui : UINT (0..10000);
END_VAR
```

If a constant is assigned to a subrange type (in the declaration or in the implementation) that does not fall into this range (e.g. 1:=5000), an error message is issued.

In order to check for observance of range boundaries at runtime, the functions **CheckRangeSigned** or **CheckRangeUnsigned** must be introduced. In these, boundary violations can be captured by the appropriate method and means (e.g. the value can be cut out or an error flag can be set.). They are implicitly called as soon as a variable is written as belonging to a subrange type constructed from either a signed or an unsigned type.

Example: In the case of a variable belonging to a signed subrange type (like i, above), the function CheckRangeSigned is called; it could be programmed as follows to trim a value to the permissible range:

```
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
   value, lower, upper: DINT;
END_VAR

IF (value < lower) THEN
   CheckRangeSigned := lower;
ELSIF(value > upper) THEN
   CheckRangeSigned := upper;
ELSE
   CheckRangeSigned := value;
END_IF
```

In calling up the function automatically, the function name CheckRangeSigned is obligatory, as is the interface specification: return value and three parameters of type DINT

When called, the function is parameterized as follows:

- value: the value to be assigned to the range type
- lower: the lower boundary of the range
- upper: the upper boundary of the range
- Return value: this is the value that is actually assigned to the range type

An assignment i:=10*y implicitly produces the following in this example:

```
i := CheckRangeSigned(10*y, -4095, 4095);
```

Even if y for example has the value 1000, then i still has only the value 4095 after this assignment.

The same applies to the function CheckRangeUnsigned: function name and interface must be correct.

```
FUNCTION CheckRangeUnsigned : UDINT
VAR_INPUT
   value, lower, upper: UDINT;
END_VAR
```

Important: If neither of the functions CheckRangeSigned or CheckRangeUnsigned is present, no type checking of subrange types occurs during runtime! The variable i could then take on any value between –32768 and 32767 at any time!

Achtung: If neither of the functions CheckRangeSigned or CheckRangeUnsigned is present like described above, there can result an endless loop if a subrange type is used in a **FOR** loop. This will happen when the range given for the FOR loop is as big or bigger than the range of the subrange type !

Example:

```
VAR
   ui : UINT (0..10000);
END_VAR

FOR ui:=0 TO 10000 DO
...
END_FOR
```

The FOR loop will never be finished, because ui cannot get bigger than 10000.

Also take care of the definition of the CheckRange functions when you define the incremental value of a FOR loop !

Appendix C: The IEC Operators and additional, norm-extending functions

CoDeSys supports all IEC operators. In contrast with the standard functions (see Appendix Appendix D: , Standard library), these operators are recognized implicitly throughout the project.

Besides the IEC operators **CoDeSys also supports the following operators which are not prescribed by the standard**: INDEXOF and SIZEOF (see Arithmetic Operators), ADR and BITADR (see Adress Operators).

Operators are used like functions in POUs.

- Arithmetic operators
- Bitstring Operators
- Bit-Shift Operators
- Selection Operators
- Comparison Operators
- Address Operators
- Calling Operator
- Type conversions
- Numeric Operators

Attention:

At operations with floating point variables the result depends on the currently used target system !

Arithmetic Operators

ADD

Addition of variables of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

Two TIME variables can also be added together resulting in another time (e.g., t#45s + t#50s = t#1m35s)

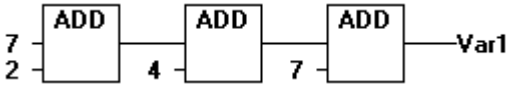
Example in IL:

LD	7	
ADD	2,4,7	
ST	Var 1	

Example in ST:

```
var1 := 7+2+4+7;
```

Example in FBD:



MUL

Multiplication of variables of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

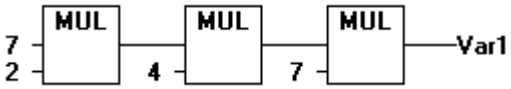
Example in IL:

```
LD      7
MUL     2,4,7
ST      Var 1
```

Example in ST:

```
var1 := 7*2*4*7;
```

Example in FBD:



SUB

Subtraction of one variable from another of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

A TIME variable may also be subtracted from another TIME variable resulting in third TIME type variable. Note that negative TIME values are undefined.

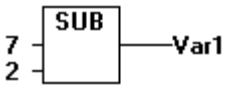
Example in IL:

```
LD      7
SUB     8
ST      Var 1
```

Example in ST:

```
var1 := 7-2;
```

Example in FBD:



DIV

Division of one variable by another of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

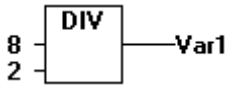
Example in IL:

```
LD      8
DIV     2
ST      Var 1           (* Result is 4 *)
```

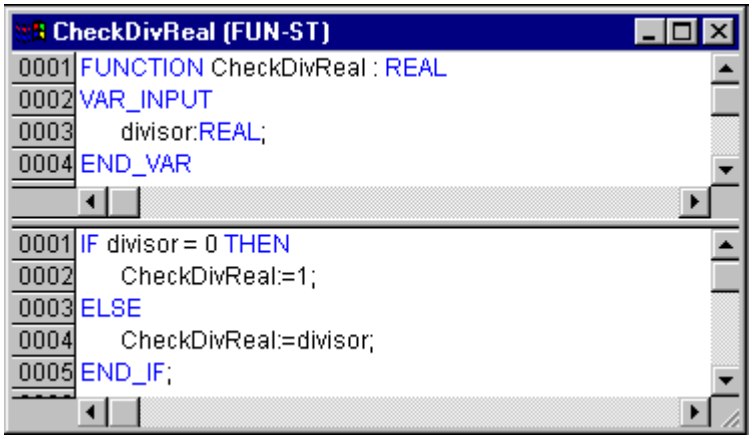
Example in ST:

```
var1 := 8/2;
```

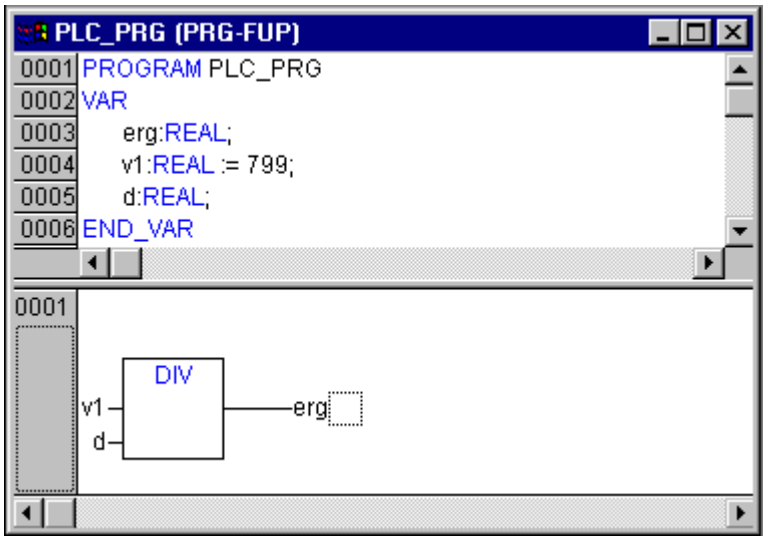
Example in FBD:



Note: If you define functions in your project with the names **CheckDivByte**, **CheckDivWord**, **CheckDivDWord** and **CheckDivReal**, you can use them to check the value of the divisor if you use the operator DIV, for example to avoid a division by 0. The functions must have the above listed names. See in the following an example for the implementation of function CheckDivReal:



Operator DIV uses the output of function CheckDivReal as divisor. In a program like shown in the following example this avoids a division by 0, the divisor (d) is set from 0 to 1. So the result of the division is 799.



MOD

Modulo Division of one variable by another of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT. The result of this function will be the remainder of the division. This result will be a whole number.

Example in IL:

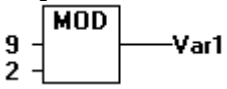
```
LD      9
MOD     2
```

```
ST      Var 1          (* Result is 1 *)
```

Example in ST:

```
var1 := 9 MOD 2;
```

Example in FBD:

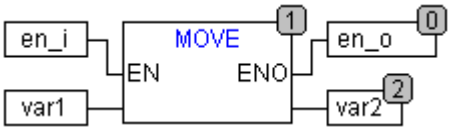


MOVE

Assignment of a variable to another variable of an appropriate type. As MOVE is available as a box in the graphic editors FBD, LD, CFC, there the (unlocking) EN/EN0 functionality can also be applied on a variable assignment.

Example in CFC in conjunction with the EN/EN0 function:

Only if en_i is TRUE, var1 will be assigned to var2.



Example in IL:

```
LD      ivar1
MOVE    ivar2
ST      ivar2          (* result: var2 gets value of var1 *)
```

```
( ! you get the same result with:
LD      ivar1
ST      ivar2 )
```

Example in ST:

```
ivar2 := MOVE(ivar1);
```

(! you get the same result with: ivar2 := ivar1;)

INDEXOF

This function is not prescribed by the standard IEC61131-3.

Perform this function to find the internal index for a POU.

Example in ST:

```
var1 := INDEXOF(POU2);
```

SIZEOF

This function is not prescribed by the standard IEC61131-3.

Perform this function to determine the number of bytes required by the given data type.

Example in IL:

```
arr1:ARRAY[0..4] OF INT;
Var1    INT
LD      arr1
```

```
SIZEOF
ST      Var 1          (* Result is 10 *)
```

Example in ST:

```
var1 := SIZEOF(arr1);
```

Bitstring Operators

AND

Bitwise AND of bit operands. The operands should be of the type BOOL, BYTE, WORD or DWORD.

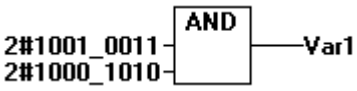
Example in IL:

```
Var1    BYTE
LD      2#1001_0011
AND     2#1000_1010
ST      Var 1          (* Result is 2#1000_0010 *)
```

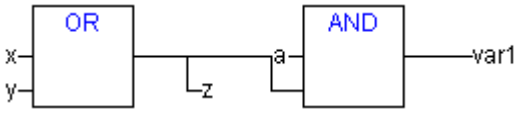
Example in ST:

```
var1 := 2#1001_0011 AND 2#1000_1010
```

Example in FBD:



Note: If you have a program step in the SFC like the following



and if you use 68xxx- or C-code generators, please note the following: The allocation of the value of the second input variable at the AND operator module to variable z will not be executed ! This is due to the optmized processing in the SFC in case of value FALSE at the input variable.

OR

Bitwise OR of bit operands. The operands should be of the type BOOL, BYTE, WORD or DWORD.

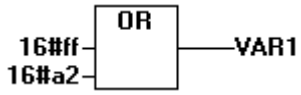
Example in IL:

```
var1 :BYTE;
LD      2#1001_0011
OR      2#1000_1010
ST      var1 (* Result is 2#1001_1011 *)
```

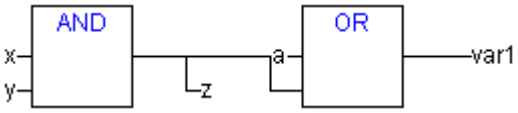
Example in ST:

```
Var1 := 2#1001_0011 OR 2#1000_1010
```

Example in FBD:



Note: If you have a program step in the SFC like the following



and if you use 68xxx- or C-code generators, please note the following: The allocation of the value of the second input variable at the AND operator module to variable z will not be executed ! This is due to the optmized processing in the SFC in case of value FALSE at the input variable.

XOR

Bitwise XOR of bit operands. The operands should be of the type BOOL, BYTE, WORD or DWORD.

Note: Regard the behaviour of the XOR function in extended form, that means if there are more than 2 inputs. The inputs will be checked in pairs and the particular results will then be compared again in pairs (this complies with the standard, but may not be expected by the user).

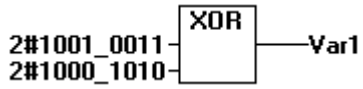
Example in IL:

```
Var1 :BYTE;
LD      2#1001_0011
XOR     2#1000_1010
ST      Var1 (* Result is 2#0001_1001 *)
```

Example in ST:

```
Var1 := 2#1001_0011 XOR 2#1000_1010
```

Example in FBD:



NOT

Bitwise NOT of a bit operand. The operand should be of the type BOOL, BYTE, WORD or DWORD.

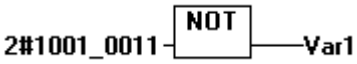
Example in IL:

```
Var1 :BYTE;
LD      2#1001_0011
NOT
ST      Var1 (* Result is 2#0110_1100 *)
```

Example in ST:

```
Var1 := NOT 2#1001_0011
```

Example in FBD:



Bit-Shift Operators

SHL

Bitwise left-shift of an operand : erg:= SHL (in, n)

in will be shifted by n bits to the left. From the right side it will be filled up with zeros.

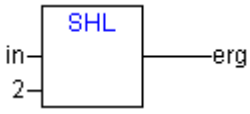
Note: Please note, that the amount of bits, which is regarded for the arithmetic operation, is pretended by the data type of the input variable !. If the input variable is a constant the smallest possible data type is regarded. The data type of the output variable has no effect at all on the arithmetic operation.

See in the following example in hexadecimal notation that you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

Example in ST:

```
shl_st (PRG-ST)
0001 PROGRAM shl_st
0002 VAR
0003     in_byte:BYTE:=16#45;
0004     in_word:WORD:=16#45;
0005     erg_byte: BYTE;
0006     erg_word: WORD;
0007     n:BYTE:=2;
0008 END_VAR

0001
0002 erg_byte:=SHL (in_byte,n);    (* Ergebnis ist 16#14*)
0003 erg_word:=SHL (in_word,n); (* Ergebnis ist 16#0114 *)
0004
```

Example in FBD:



Example in IL:

```
LD      16#45
SHL     2
ST      erg_byte
```

SHR

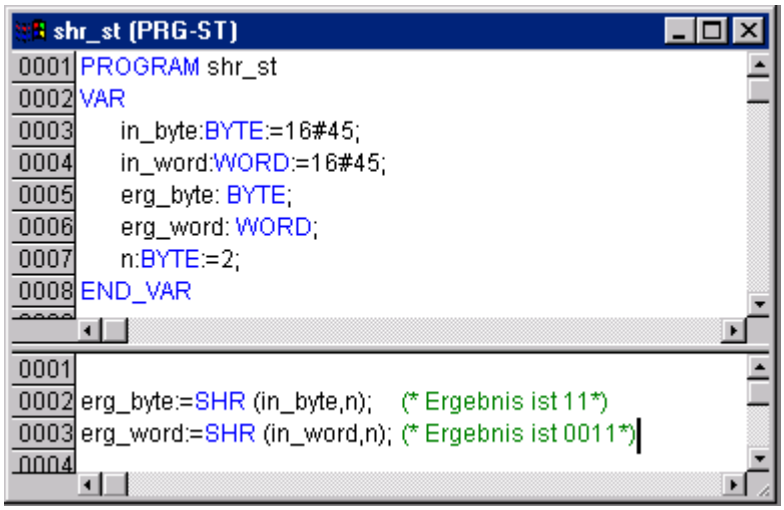
Bitwise right-shift of an operand: erg:= SHR (in, n)

in will be shifted to the right by n bits. If an signed data type is used, then an arithmetic shift will be done, that means that from the left it will be filled up with '0' or with '1', depending on which value the uppermost bit of in has. For BYTE, WORD and DWORD it will be filled up with zeros.

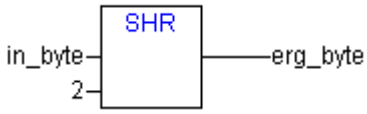
Note: Please note, that the amount of bits, which is regarded for the arithmetic operation, is pretended by the data type of the input variable !. If the input variable is a constant the smallest possible data type is regarded. The data type of the output variable has no effect at all on the arithmetic operation.

See the following example in hexadecimal notation to notice the results of the arithmetic operation depending on the type of the input variable (BYTE or WORD).

Example in ST:



Example in FBD:



Example in IL:

```
LD      16#45
SHL     2
ST      erg_byte
```

ROL

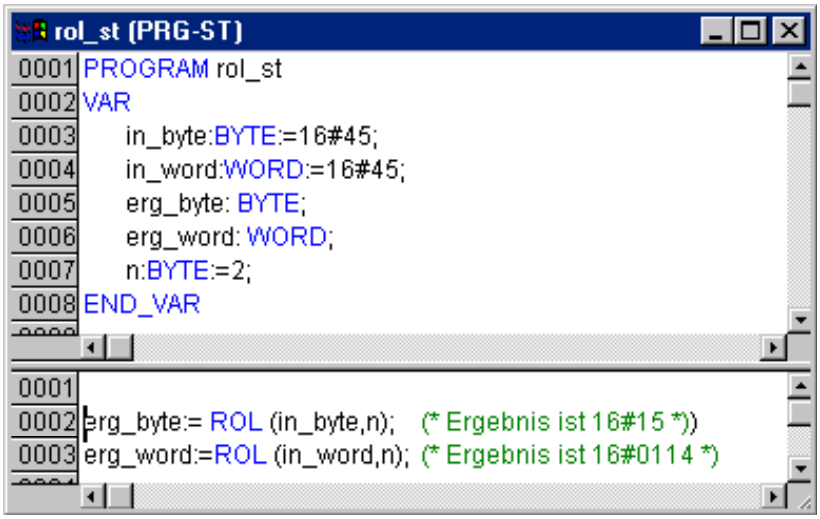
Bitwise rotation of an operand to the left: erg:= ROL (in, n)

erg, in and n should be of the type BYTE, WORD or DWORD. in will be shifted one bit position to the left n times while the bit that is furthest to the left will be reinserted from the right.

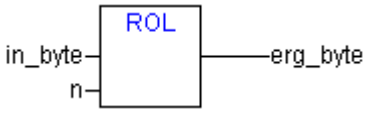
Note: Please note, that the amount of bits, which is regarded for the arithmetic operation, is pretended by the data type of the input variable !. If the input variable is a constant the smallest possible data type is regarded. The data type of the output variable has no effect at all on the arithmetic operation.

See in the following example in hexadecimal notation that you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

Example in ST:



Example in FBD:



Example in IL:

```
LD      16#45
SHL     2
ST      erg_byte
```

ROR

Bitwise rotation of an operand to the right: erg = ROR (in, n)

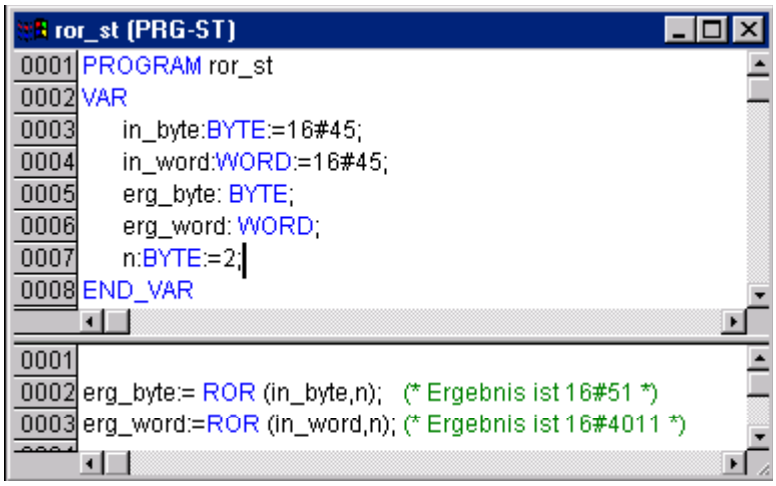
erg, in and n should be of the type BYTE, WORD or DWORD. in will be shifted one bit position to the right n times while the bit that is furthest to the left will be reinserted from the left.

Note: Please note, that the amount of bits, which is regarded for the arithmetic operation, is pretended by the data type of the input variable !. If the input variable is a constant the smallest possible data type is regarded. The data type of the output variable has no effect at all on the arithmetic operation.

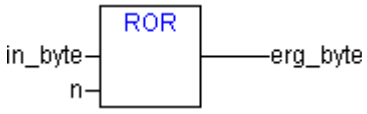
See in the following example in hexadecimal notation that you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE

or WORD), although the values of the input variables in_byte and in_word are the same.

Example in ST:



Example in FBD:



Example in IL:

```
LD      16#45
SHL     2
ST      erg_byte
```

Selection Operators

All selection operations can also be performed with variables. For purposes of clarity we will limit our examples to the following which use constants as operators.

SEL

Binary Selection.
OUT := SEL(G, IN0, IN1) means:
OUT := IN0 if G=FALSE;
OUT := IN1 if G=TRUE.

IN0, IN1 and OUT can be any type of variable, G must be BOOL. The result of the selection is IN0 if G is FALSE, IN1 if G is TRUE.

Example in IL:

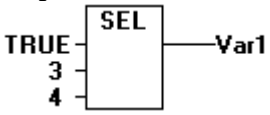
```
LD      TRUE
SEL     3,4
ST      Var1            (* Result ist 4 *)

LD      FALSE
SEL     3,4
```

```
ST      Var1          (* Result ist 3 *)
```

Example in ST:

```
Var1:=SEL(TRUE,3,4);   (* Result is 4 *)
```

Example in FBD:



Note: Note that an expression occurring ahead of IN1 or IN2 will not be processed if IN0 is TRUE.

MAX

Maximum function. Returns the greater of the two values.

```
OUT := MAX(IN0, IN1)
```

IN0, IN1 and OUT can be any type of variable.

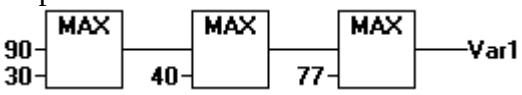
Example in IL:

```
LD      90
MAX     30
MAX     40
MAX     77
ST      Var1          (* Result is 90 *)
```

Example in ST:

```
Var1:=MAX(30,40);   (* Result is 40 *)
Var1:=MAX(40,MAX(90,30));   (* Result is 90 *)
```

Example in FBD:



MIN

Minimum function. Returns the lesser of the two values.

```
OUT := MIN(IN0, IN1)
```

IN0, IN1 and OUT can be any type of variable.

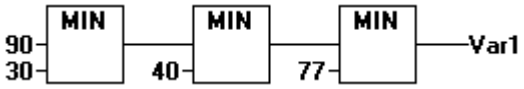
Example in IL:

```
LD      90
MIN     30
MIN     40
MIN     77
ST      Var 1         (* Result is 30 *)
```

Example in ST:

```
Var1:=MIN(90,30);  (* Result is 30 *);
Var1:=MIN(MIN(90,30),40);  (* Result is 30 *);
```

Example in FBD:



LIMIT

Limiting
OUT := LIMIT(Min, IN, Max) means:
OUT := MIN (MAX (IN, Min), Max)

Max is the upper and Min the lower limit for the result. Should the value IN exceed the upper limit Max, LIMIT will return Max. Should IN fall below Min, the result will be Min.

IN and OUT can be any type of variable.

Example in IL:

```
LD      90
LIMIT   30,80
ST      Var 1          (*Result is 80 *)
```

Example in ST:

```
Var1:=LIMIT(30,90,80);  (* Result is 80 *);
```

MUX

Multiplexer
OUT := MUX(K, IN0,...,INn) means:
OUT := IN_K.

IN0, ...,INn and OUT can be any type of variable. K must be BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT or UDINT. MUX selects the Kth value from among a group of values.

Example in IL:

```
LD      0
MUX     30,40,50,60,70,80
ST      Var 1              (*Result is 30 *)
```

Example in ST:

```
Var1:=MUX(0,30,40,50,60,70,80);  (* Result is 30 *);
```

Note: Note that an expression occurring ahead of an input other than IN_K will not be processed to save run time ! Only in simulation mode all expressions will be executed.

Comparison Operators

GT

Greater than

A Boolean operator which returns the value TRUE when the value of the first operand is greater than that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

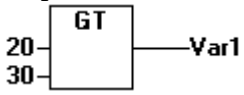
Example in IL:

```
LD      20
GT      30
ST      Var 1          (* Result is FALSE *)
```

Example in ST:

```
VAR1 := 20 > 30 > 40 > 50 > 60 > 70;
```

Example in FBD:



LT

Less than

A Boolean operator that returns the value TRUE when the value of the first operand is less than that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

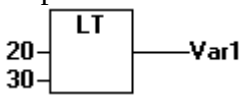
Example in IL:

```
LD      20
LT      30
ST      Var 1          (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 20 < 30;
```

Example in FBD:



LE

Less than or equal to

A Boolean operator that returns the value TRUE when the value of the first operand is less than or equal to that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

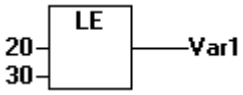
Example in IL:

```
LD      20
LE      30
ST      Var 1       (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 20 <= 30;
```

Example in FBD



GE

Greater than or equal to

A Boolean operator that returns the value TRUE when the value of the first operand is greater than or equal to that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

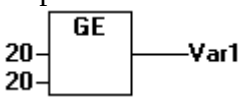
Example in IL:

```
LD      60
GE      40
ST      Var 1       (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 60 >= 40;
```

Example in FBD:



EQ

Equal to

A Boolean operator that returns the value TRUE when the operands are equal. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

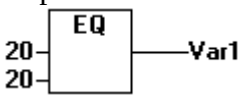
Example in IL:

```
LD      40
EQ      40
ST      Var 1       (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 40 = 40;
```

Example in FBD:



NE

Not equal to

A Boolean operator that returns that value TRUE when the operands are not equal. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

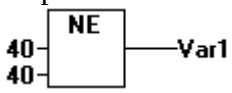
Example in IL:

```
LD      40
NE      40
ST      Var 1          (* Result is FALSE *)
```

Example in ST:

```
VAR1 := 40 <> 40;
```

Example in FBD:



Address Operators

ADR

Address Function, not prescribed by the standard IEC61131-3.

ADR returns the address of its argument in a DWORD. This address can be sent to manufacturing functions to be treated as a pointer or it can be assigned to a pointer within the project.

dwVar:=ADR(bVAR);

Example in IL:

```
LD          Var 1
ADR
ST          Var 2
man_fun1
```

BITADR

Adress function, not prescribed by the standard IEC61131-3.

BITADR returns the bit offset within the segment in a DWORD. Regard that the offset value depends on whether the option byte addressing in the target settings is activated or not.

```
VAR
    var1 AT %IX2.3:BOOL;
    bitoffset: DWORD;
END_VAR
```

Example in ST:

```
bitoffset:=BITADR(var1); (* Result if byte addressing=TRUE: 19, if byte
addressing=FALSE: 35 *)
```

Example in AWL:

```
LD          Var1
BITADR
ST          Var2
```

Content Operator

A pointer can be dereferenced by adding the content operator "^" after the pointer identifier.

Example in ST:

```
pt:POINTER TO INT;
var_int1:INT;
var_int2:INT;
pt := ADR(var_int1);
var_int2:=pt^;
```

Calling Operator

CAL

Calling a function block or a program

Use CAL in IL to call up a function block instance. The variables that will serve as the input variables are placed in parentheses right after the name of the function block instance.

Example: Calling up the instance *Inst* from a function block where input variables *Par1* and *Par2* are 0 and TRUE respectively.

```
CAL INST(PAR1 := 0, PAR2 := TRUE)
```

Type Conversion Functions

Its is forbidden to implicitly convert from a ”larger” type to a ”smaller” type (for example from INT to BYTE or from DINT to WORD). Special type conversions are required if one wants to do this. One can basically convert from any elementary type to any other elementary type.

Syntax:

```
<elem.Typ1>_TO_<elem.Typ2>
```

Please regard that at ...TO_STRING conversions the string is generated left-justified. If it is defined to short, it will be cut from the right side.

BOOL_TO Conversions

Conversion from type BOOL to any other type:

For number types the result is 1, when the operand is TRUE, and 0, when the operand is FALSE.

For the STRING type the result is ‚TRUE‘ or ‚FALSE‘.

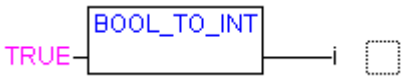
Examples in AWL:

```
LD      TRUE                    (*Result is 1 *)
BOOL_TO_INT
ST      i

LD      TRUE                    (*Result is 'TRUE' *)
BOOL_TO_STRING
ST      str

LD      TRUE                    (*Result is T#1ms *)
BOOL_TO_TIME
ST      t

LD      TRUE                    (*Result is TOD#00:00:00.001 *)
BOOL_TO_TOD
ST

LD      FALSE                   (*Result is D#1970-01-01 *)
BOOL_TO_DATE
ST      dat

LD      TRUE                    (*Result is DT#1970-01-01-00:00:01 *)
BOOL_TO_DT
ST      dandt
```

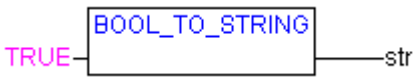
Examples in St:

```
i:=BOOL_TO_INT(TRUE);           (* Result is 1 *)
str:=BOOL_TO_STRING(TRUE);      (* Result is "TRUE" *)
t:=BOOL_TO_TIME(TRUE);          (* Result is T#1ms *)
tof:=BOOL_TO_TOD(TRUE);         (* Result is TOD#00:00:00.001 *)
dat:=BOOL_TO_DATE(FALSE);       (* Result is D#1970 *)
dandt:=BOOL_TO_DT(TRUE);        (* Result is
                                DT#1970-01-01-00:00:01 *)
```

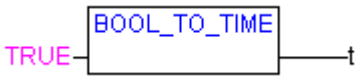
Examples in FUP:



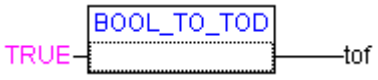
(*Result is 1 *)



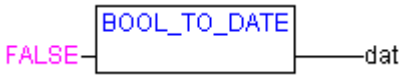
(*Result is 'TRUE' *)



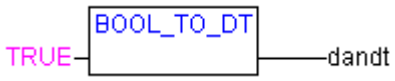
(*Result is T#1ms *)



(*Result is TOD#00:00:00.001 *)



(*Result is D#1970-01-01 *)



(*Result is DT#1970-01-01-00:00:01 *)

TO_BOOL Conversions

Conversion from another variable type to BOOL:

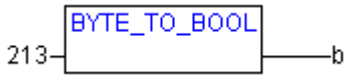
The result is TRUE when the operand is not equal to 0. The result is FALSE when the operand is equal to 0.

The result is true for STRING type variables when the operand is "TRUE", otherwise the result is FALSE.

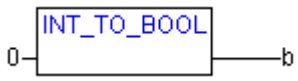
Examples in AWL:

```
LD      213                     (*Result is TRUE *)
BYTE_TO_BOOL
ST      b

LD      0                       (*Result is FALSE *)
INT_TO_BOOL
ST      b

LD      T#5ms                   (*Result is TRUE *)
TIME_TO_BOOL
ST      b

LD      'TRUE'                  (*Result is TRUE *)
STRING_TO_BOOL
ST      b
```

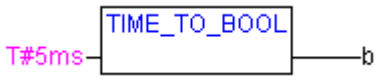
Examples in FUP:



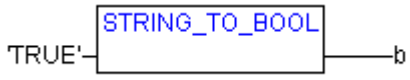
(*Result is TRUE *)



(*Result is FALSE *)



(*Result is TRUE *)



(*Result is TRUE *)

Examples in St:

```
b := BYTE_TO_BOOL(2#11010101);   (* Result is TRUE *)
b := INT_TO_BOOL(0);             (* Result is FALSE *)
b := TIME_TO_BOOL(T#5ms);        (* Result is TRUE *)
b := STRING_TO_BOOL('TRUE');     (* Result is TRUE *)
```

Conversion between Integral Number Types

Conversion from an integral number type to another number type:

When you perform a type conversion from a larger to a smaller type, you risk losing some information. If the number you are converting exceeds the range limit, the first bytes for the number will be ignored.

Example in ST:

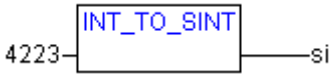
```
si := INT_TO_SINT(4223); (* Result is 127 *)
```

If you save the integer 4223 (16#107f represented hexadecimally) as a SINT variable, it will appear as 127 (16#7f represented hexadecimally).

Example in IL:

```
LD          2
INT_TO_REAL
MUL         3.5
```

Example in FBD:



REAL_TO-/ LREAL_TO Conversions

Converting from the variable type REAL or LREAL to a different type:

The value will be rounded up or down to the nearest whole number and converted into the new variable type. Exceptions to this are the variable types STRING, BOOL, REAL and LREAL.

Please regard at a conversion to type STRING that the total number of digits is limited to 16. If the (L)REAL-number has more digits, then the sixteenth will be rounded. If the length of the STRING is defined to short, it will be cut beginning from the right end.

When you perform a type conversion from a larger to a smaller type, you risk losing some information.

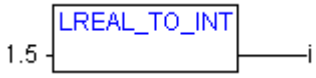
Example in ST:

```
i := REAL_TO_INT(1.5); (* Result is 2 *)
j := REAL_TO_INT(1.4); (* Result is 1 *)
i := REAL_TO_INT(-1.5); (* result is -2 *)
j := REAL_TO_INT(-1.4); (* result is -1 *)
```

Example in IL:

```
LD          2.7
REAL_TO_INT
GE          %MW8
```

Example in FBD:



TIME_TO/TIME_OF_DAY Conversions

Converting from the variable type TIME or TIME_OF_DAY to a different type:

The time will be stored internally in a DWORD in milliseconds (beginning with 12:00 A.M. for the TIME_OF_DAY variable). This value will then be converted.

When you perform a type conversion from a larger to a smaller type, you risk losing some information

For the STRING type variable, the result is a time constant.

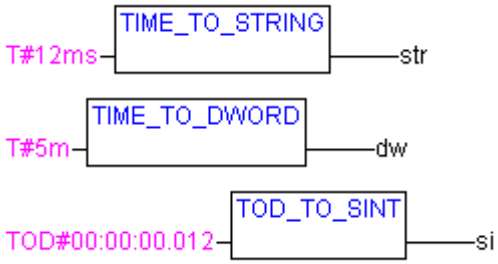
Examples in IL:

```
LD      T#12ms                  (*Result is 'T#12ms' *)
        TIME_TO_STRING
ST      str
LD      T#300000ms              (*Result is 300000 *)
TIME_TO_DWORD
ST      dw
LD      TOD#00:00:00.012        (*Result is 12 *)
TOD_TO_SINT
ST      si
```

Examples in St:

```
str :=TIME_TO_STRING(T#12ms);
dw:=TIME_TO_DWORD(T#5m);
si:=TOD_TO_SINT(TOD#00:00:00.012);
```

Examples in FBD:



DATE_TO/DT_TO Conversions

Converting from the variable type DATE or DATE_AND_TIME to a different type:

The date will be stored internally in a DWORD in seconds since Jan. 1, 1970. This value will then be converted.

When you perform a type conversion from a larger to a smaller type, you risk losing some information

For STRING type variables, the result is the date constant.

Examples in St:

```
b :=DATE_TO_BOOL(D#1970-01-01);              (* Result is FALSE *)
i :=DATE_TO_INT(D#1970-01-15);               (* Result is 29952 *)
byt :=DT_TO_BYTE(DT#1970-01-15-05:05:05);    (* Result is 129 *)
str:=DT_TO_STRING(DT#1998-02-13-14:20);      (* Result is
                                             'DT#1998-02-13-14:20' *)
```

STRING_TO Conversions

Converting from the variable type STRING to a different type:

The operand from the STRING type variable must contain a value that is valid in the target variable type, otherwise the result will be 0.

Examples in St:

```
b :=STRING_TO_BOOL('TRUE');      (* Result is TRUE *)
w :=STRING_TO_WORD('abc34');     (* Result is 0 *)
t :=STRING_TO_TIME('T#127ms');   (* Result is T#127ms *)
```

TRUNC

Converting from REAL to INT. The whole number portion of the value will be used.

When you perform a type conversion from a larger to a smaller type, you risk losing some information.

Examples in ST:

```
i:=TRUNC(1.9); (* Result is 1 *)
i:=TRUNC(-1.4); (* result is -1 *).
```

Example in IL:

```
LD      2.7
TRUNC
GE      %MW8
```

Numeric Functions

ABS

Returns the absolute value of a number. ABS(-2) equals 2.

The following type combinations for input and output variables are possible:

IN	OUT
INT	INT, REAL, WORD, DWORD, DINT
REAL	REAL
BYTE	INT, REAL, BYTE, WORD, DWORD, DINT
WORD	INT, REAL, WORD, DWORD, DINT
DWORD	REAL, DWORD, DINT
SINT	REAL
USINT	REAL
UINT	INT, REAL, WORD, DWORD, DINT, UDINT, UINT
DINT	REAL, DWORD, DINT
UDINT	REAL, DWORD, DINT, UDINT

Example in IL:

```
LD      2
ABS
ST      i       (*Result is 2 *)
```

Example in ST:

```
i:=ABS(-2);
```

Example in FBD:

SQRT

Returns the square root of a number.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD      16
SQRT
ST      q       (*Result is 4 *)
```

Example in ST:

```
q:=SQRT(16);
```

Example in FBD:

LN

Returns the natural logarithm of a number.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

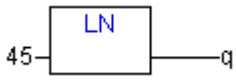
Example in IL:

```
LD      45
LN
ST      q       (*Result is 3.80666 *)
```

Example in ST:

```
q:=LN(45);
```

Example in FBD:



LOG

Returns the logarithm of a number in base 10.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD      314.5
LOG
ST      q       (*Result is 2.49762 *)
```

Example in ST:

```
q:=LOG(314.5);
```

Example in FBD:

EXP

Returns the exponential function.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD      2
EXP
ST      q       (* result is 7.389056099 *)
```

Example in ST:

```
q:=EXP(2);
```

Example in FBD:

SIN

Returns the sine of a number.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD      0.5
SIN
ST      q       (*Result is 0.479426 *)
```

Example in ST:

```
q:=SIN(0.5);
```

Example in FBD:

COS

Returns the cosine of number. The value is calculated in arch minutes.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type Typ REAL.

Example in IL:

```
LD      0.5
COS
ST      q       (*Result is 0.877583 *)
```

Example in ST:q:=COS(0.5);

Example in FBD:

TAN

Returns the tangent of a number. The value is calculated in arch minutes. IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD      0.5
TAN
ST      q       (*Result is 0.546302 *)
```

Example in ST:

```
q:=TAN(0.5);
```

Example in FBD:

ASIN

Returns the arc sine (inverse function of sine) of a number. .

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD      0.5
ASIN
ST      q       (*Result is 0.523599 *)
```

Example in ST:

```
q:=ASIN(0.5);
```

Example in FBD:

ACOS

Returns the arc cosine (inverse function of cosine) of a number. The value is calculated in arch minutes.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

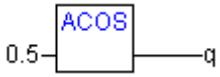
Example in IL:

```
LD      0.5
ABS
ST      q       (*Result is 1.0472 *)
```

Example in ST:

```
q:=ACOS(0.5);
```

Example in FBD:



ATAN

Returns the arc tangent (inverse function of tangent) of a number. The value is calculated in arch minutes.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD      0.5
ABS
ST      q       (*Result is 0.463648 *)
```

Example in ST:

```
q:=ATAN(0.5);
```

Example in FBD:

EXPT

Exponentiation of a variable with another variable:
OUT = $IN1^{IN2}$.

IN1 and IN2 can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

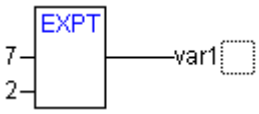
Example in IL:

```
LD      7
EXPT    2
ST      var1        (*Result is 49 *)
```

Example in ST:

```
var1 := EXPT(7,2);
```

Example in FBD:



Appendix D: Standard Library Elements

String functions

Please note:

String functions are not "thread safe": When using tasks, string functions may only be used in a single task. If the same function is used in different tasks, there is a danger of overwriting.

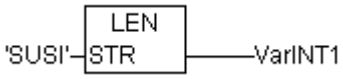
LEN

Returns the length of a string. Input STR is of type STRING, the return value of the function is type INT.

Example in IL:

```
LD      'SUSI'
LEN
ST      VarINT1    (* Ergebnis ist 4 *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := LEN ('SUSI');
```

LEFT

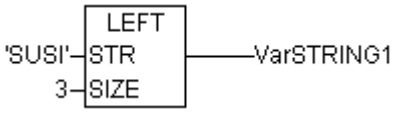
Left returns the left, initial string for a given string. Input STR is type STRING, SIZE is of type INT, the return value of the function is type STRING.

LEFT (STR, SIZE) means: Take the first SIZE character from the right in the string STR.

Example in IL:

```
LD      'SUSI'
LEFT    3
ST      VarSTRING1    (* Ergebnis ist 'SUS' *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := LEFT ('SUSI',3);
```

RIGHT

Right returns the right, initial string for a given string.

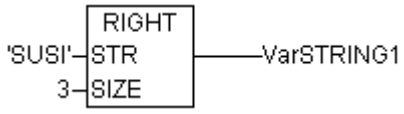
RIGHT (STR, SIZE) means: Take the first SIZE character from the right in the string STR.

Input STR is of type STRING, SIZE is of type INT, the return value of the function is of type STRING.

Example in IL:

```
LD      'SUSI'
RIGHT   3
ST      VarSTRING1   (* Ergebnis ist ‘USI’ *)
```

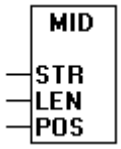
Example in FBD:



Example in ST:

```
VarSTRING1 := RIGHT (‘SUSI’,3);
```

MID



Mid returns a partial string from within a string.

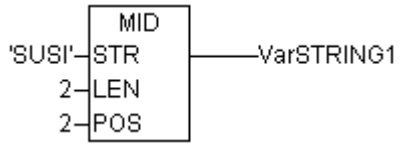
Input STR is type STRING, LEN and POS are type INT, the return value of the function is type STRING.

MID (STR, LEN, POS) means: Retrieve LEN characters from the STR string beginning with the character at position POS.

Example in IL:

```
LD      'SUSI'
RIGHT   2,2
ST      VarSTRING1   (* Ergebnis ist ‘US’ *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := MID (‘SUSI’,2,2);
```

CONCAT

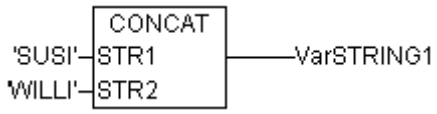
Concatenation (combination) of two strings.

The input variables STR1 and STR2 as well as the return value of the function are type STRING.

Example in IL:

```
LD      'SUSI'
CONCAT  ‘WILLI’
ST      VarSTRING1    (* Ergebnis ist ‘SUSIWILLI’ *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := CONCAT (‘SUSI’,’WILLI’);
```

INSERT

INSERT inserts a string into another string at a defined point.

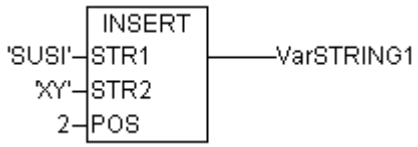
The input variables STR1 and STR2 are type STRING, POS is type INT and the return value of the function is type STRING.

INSERT(STR1, STR2, POS) means: insert STR2 into STR1 after position POS.

Example in IL:

```
LD      'SUSI'
INSERT  ‘XY’,2
ST      VarSTRING1    (* Ergebnis ist ‘SUXYSI’ *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := INSERT (‘SUSI’,’XY’,2);
```

DELETE

DELETE removes a partial string from a larger string at a defined position.

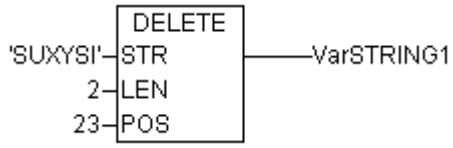
The input variable STR is type STRING, LEN and POS are type INT, the return value of the function is type STRING.

DELETE(STR, L, P) means: Delete L characters from STR beginning with the character in the P position.

Example in IL:

```
LD      'SUXYSI'
DELETE  2,23
ST      Var1        (* Ergebnis ist ‘SUSI’ *)
```

Example in FBD:



Example in ST:

```
Var1 := DELETE (‘SUXYSI’,2,3);
```

REPLACE

REPLACE replaces a partial string from a larger string with a third string.

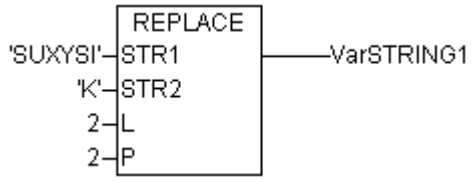
The input variable STR1 and STR2 are type STRING, LEN and POS are type INT, the return value of the function is type STRING.

REPLACE(STR1, STR2, L, P) means: Replace L characters from STR1 with STR2 beginning with the character in the P position.

Example in IL:

```
LD      'SUXYSI'
REPLACE 'K', 2,2
ST      VarSTRING1  (* Ergebnis ist ‘SKYSI’ *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := REPLACE (‘SUXYSI’,’K’,2,2);
```

FIND

FIND searches for a partial string within a string.

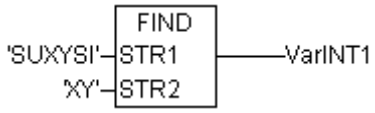
The input variable STR1 and STR2 are type STRING, the return value of the function is type STRING.

FIND(STR1, STR2) means: Find the position of the first character where STR2 appears in STR1 for the first time. If STR2 is not found in STR1, then OUT:=0.

Example in IL:

```
LD      'SUXYSI'
FIND    'XY'
ST      VarINT1     (* Ergebnis ist ‘3’ *)
```

Example in FBD:



Example in ST:

```
arINT1 := FIND ('SUXYSI','XY');
```

Bistable Function Blocks

SR

Making Bistable Function Blocks Dominant:

Q1 = SR (SET1, RESET) means:
Q1 = (NOT RESET AND Q1) OR SET1

The input variables SET1 and RESET as well as the output variable Q1 are type BOOL.

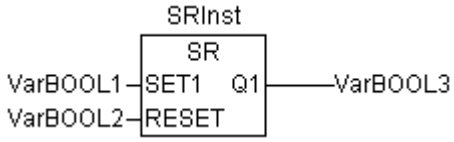
Declaration example:

```
SRInst : SR ;
```

Example in IL:

```
CAL     SRInst(SET1 := VarBOOL1, RESET := VarBOOL2)
LD      SRInst.Q1
ST      VarBOOL3
```

Example in FBD:



Example in ST:

```
SRInst(SET1:= VarBOOL1 , RESET:=VarBOOL2 );
VarBOOL3 := SRInst.Q1 ;
```

RS

Resetting Bistable Function Blocks

Q1 = RS (SET, RESET1) means:
Q1 = NOT RESET1 AND (Q1 OR SET)

The input variables SET and RESET1 as well as the output variable Q1 are type BOOL.

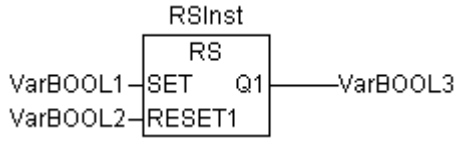
Declaration example:

```
RSInst : RS ;
```

Example in IL:

```
CAL     RSInst(SET := VarBOOL1, RESET1 := VarBOOL2)
LD      RSInst.Q1
ST      VarBOOL3
```

Example in FBD:



Example in ST:

```
RSInst(SET:= VarBOOL1 , RESET1:=VarBOOL2 );
VarBOOL3 := RSInst.Q1 ;
```

SEMA

A Software Semaphore (Interruptible)

BUSY = SEMA(CLAIM, RELEASE) means:

```
BUSY := X;
IF CLAIM THEN X:=TRUE;
ELSE IF RELEASE THEN BUSY := FALSE; X:= FALSE;
END_IF
```

X is an internal BOOL variable that is FALSE when it is initialized. The input variables CLAIM and RELEASE as well as the output variable BUSY are type BOOL.

If BUSY is TRUE when SEMA is called up, this means that a value has already been assigned to SEMA (SEMA was called up with CLAIM = TRUE). If BUSY is FALSE, SEMA has not yet been called up or it has been released (called up with RELEASE = TRUE).

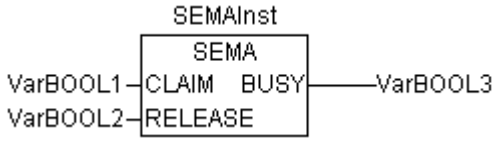
Declaration example:

```
SEMAInst : SEMA ;
```

Example in IL:

```
CAL     SEMAInst(CLAIM := VarBOOL1, RELEASE := VarBOOL2)
LD      SEMAInst.BUSY
ST      VarBOOL3
```

Example in FBD:



Example in ST:

```
SEMAInst(CLAIM:= VarBOOL1 , RELEASE:=VarBOOL2 );
VarBOOL3 := SEMAInst.BUSY;
```

Trigger

R_TRIG

The function block R_TRIG detects a rising edge.

```
FUNCTION_BLOCK R_TRIG
VAR_INPUT
   CLK : BOOL;
END_VAR
VAR_OUTPUT
   Q : BOOL;
END_VAR
VAR
   M : BOOL := FALSE;
END_VAR
   Q0 := CLK AND NOT M;
   M := CLK;
END_FUNCTION_BLOCK
```

The output Q0 and the help variable M will remain FALSE as long as the input variable CLK is FALSE. As soon as S1 returns TRUE, Q will first return TRUE, then M will be set to TRUE. This means each time the function is called up, Q will return FALSE until CLK has falling edge followed by an rising edge.

Declaration example:

```
RTRIGInst : R_TRIG ;
```

Example in IL:

```
CAL     RTRIGInst(CLK := VarBOOL1)
LD      RTRIGInst.Q
ST      VarBOOL2
```

Example in FBD:

Example in ST:

```
RTRIGInst(CLK:= VarBOOL1);
VarBOOL2 := RTRIGInst.Q;
```

F_TRIG

The function block F_TRIG a falling edge.

```
FUNCTION_BLOCK F_TRIG
VAR_INPUT
   CLK: BOOL;
END_VAR
VAR_OUTPUT
   Q: BOOL;
END_VAR
VAR
   M: BOOL := FALSE;
```

```
END_VAR
   Q := NOT CLK AND NOT M;
   M := NOT CLK;
END_FUNCTION_BLOCK
```

The output Q and the help variable M will remain FALSE as long as the input variable CLK returns TRUE. As soon as CLK returns FALSE, Q will first return TRUE, then M will be set to TRUE. This means each time the function is called up, Q will return FALSE until CLK has a rising followed by a falling edge.

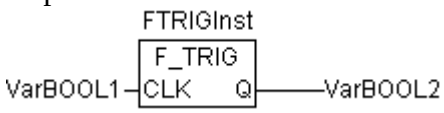
Declaration example:

```
FTRIGInst : F_TRIG ;
```

Example in IL:

```
CAL     FTRIGInst(CLK := VarBOOL1)
LD      FTRIGInst.Q
ST      VarBOOL2
```

Example in FBD:



Example in ST:

```
FTRIGInst(CLK:= VarBOOL1);
VarBOOL2 := FTRIGInst.Q;
```

Counter

CTU

The function block Incrementer:

The input variables CU and RESET as well as the output variable Q are type BOOL, the input variable PV and the output variable CV are type INT.

The counter variable CV will be initialized with 0 if RESET is TRUE. If CU has a rising edge from FALSE to TRUE, CV will be raised by 1.Q will return TRUE when CV is greater than or equal to the upper limit PV.

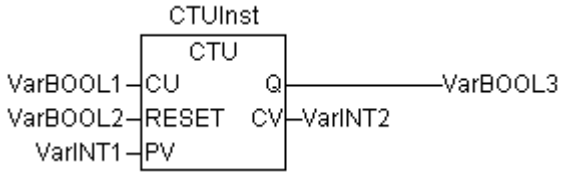
Declaration example:

```
CTUInst : CTU ;
```

Example in IL:

```
CAL     CTUInst(CU := VarBOOL1, RESET := VarBOOL2, PV := VarINT1)
LD      CTUInst.Q
ST      VarBOOL3
LD      CTUInst.CV
ST      VarINT2
```

Example in FBD:



Example in ST:

```
CTUInst(CU:= VarBOOL1, RESET:=VarBOOL2 , PV:= VarINT1);
VarBOOL3 := CTUInst.Q ;
VarINT2 := CTUInst.CV;
```

CTD

Function Block Decrementer:

The input variables CD and LOAD as well as the output variable Q are type BOOL, the input variable PV and the output variable CV are type INT.

When LOAD_ is TRUE, the counter variable CV will be initialized with the upper limit PV. If CD has a rising edge from FALSE to TRUE, CV will be lowered by 1 provided CV is greater than 0 (i.e., it doesn't cause the value to fall below 0).

Q returns TRUE when CVis equal 0.

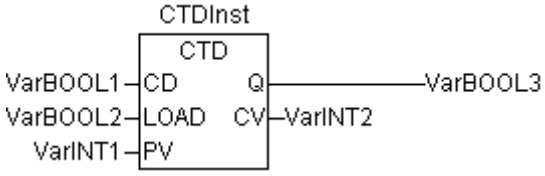
Declaration example:

```
CTDInst : CTD ;
```

Example in IL:

```
CAL     CTDInst(CD := VarBOOL1, LOAD := VarBOOL2, PV := VarINT1)
LD      CTDInst.Q
ST      VarBOOL3
LD      CTDInst.CV
ST      VarINT2
```

Example in FBD:



Example in ST:

```
CTDInst(CD:= VarBOOL1, LOAD:=VarBOOL2 , PV:= VarINT1);
VarBOOL3 := CTDInst.Q ;
VarINT2 := CTDInst.CV;
```

CTUD

Function Block Incrementer/Decrementer

The input variables CU, CD, RESET, LOAD as well as the output variables QU and QD are type BOOL, PV and CV are type INT.

If RESET is valid, the counter variable CV will be initialized with 0. If LOAD is valid, CV will be initialized with PV.

If CU has a rising edge from FALSE to TRUE, CV will be raised by 1. If CD has a rising edge from FALSE to TRUE, CV will be lowered by 1 provided this does not cause the value to fall below 0.

QU returns TRUE when CV has become greater than or equal to PV.

QD returns TRUE when CV has become equal to 0.

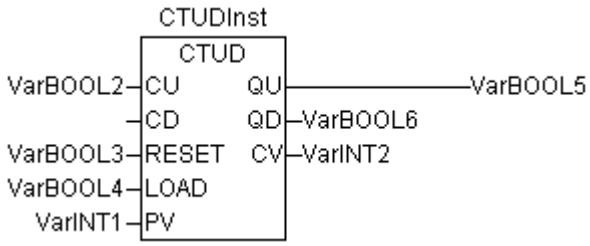
Declaration example:

```
CTUDInst : CUTD ;
```

Example in IL:

```
CAL     CTUDInst(CU := VarBOOL2, RESET := VarBOOL3, LOAD
        := VarBOOL4, PV := VarINT1)
LD      CTUDInst.QU
ST      VarBOOL5
LD      CTUDInst.QD
ST      VarBOOL6
LD      CTUDInst.CV
ST      VarINT2
```

Example in FBD:



Example in ST:

```
CTUDInst(CU := VarBOOL1, CU:= VarBOOL2, RESET := VarBOOL3,
   LOAD:=VarBOOL4 , PV:= VarINT1);
VarBOOL5 := CTUDInst.QU ;
VarBOOL6 := CTUDInst.QD ;
VarINT2 := CTUDInst.CV;
```

Timer

TP

The function blockTimer is a trigger. TP(IN, PT, Q, ET) means:

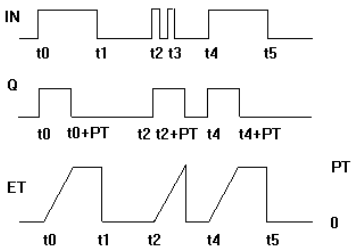
IN and PT are input variables of the BOOL and TIME types respectively. Q and ET are output variables of the BOOL and TIME types respectively. If IN is FALSE, Q is FALSE and ET is 0.

As soon as IN becomes TRUE, the time will begin to be counted in milliseconds in ET until its value is equal to PT. It will then remain constant.

Q is TRUE if IN is TRUE and ET is less than or equal to PT. Otherwise it is FALSE.

Q returns a signal for the time period given in PT.

Graphic Display of the TP Time Sequence



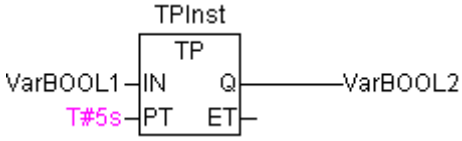
Declaration example:

```
TPInst : TP ;
```

Example in IL:

```
CAL     TPInst(IN := VarBOOL1, PT := T#5s)
LD      TPInst.Q
ST      VarBOOL2
```

Example in FBD:



Example in ST:

```
TPInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 :=TPInst.Q;
```

TON

The function block Timer On Delay implements a turn-on delay..

TON(IN, PT, Q, ET) means:

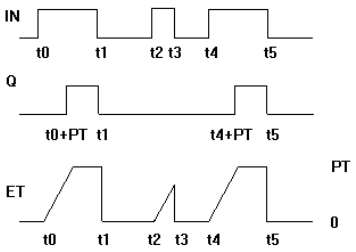
IN and PT are input variables of the BOOL and TIME types respectively. Q and ET are output variables of the BOOL and TIME types respectively. If IN is FALSE, Q is FALSE and ET is 0.

As soon as IN becomes TRUE, the time will begin to be counted in milliseconds in ET until its value is equal to PT. It will then remain constant.

Q is TRUE when IN is TRUE and ET is equal to PT. Otherwise it is FALSE.

Thus, Q has a rising edge when the time indicated in PT in milliseconds has run out.

Graphic display of TON behavior over time:



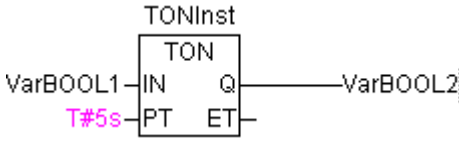
Declaration example:

```
TONInst : TON ;
```

Example in IL:

```
CAL     TONInst(IN := VarBOOL1, PT := T#5s)
LD      TONInst.Q
ST      VarBOOL2
```

Example in FBD:



Example in ST:

```
TONInst(IN := VarBOOL1, PT:= T#5s);
```

TOF

The function block TOF implements a turn-off delay..

TOF(IN, PT, Q, ET) means:

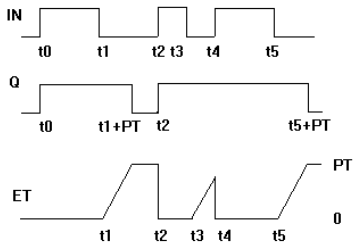
IN and PT are input variables type BOOL respectively TIME. Q and E are output variabls type BOOL respectively TIME. If IN is TRUE, the outputs are TRU respectively 0.

As soon as IN becomes FALSE, in ET the time will begin to be counted in milliseconds in ET until its value is equal to PT. It will then remain constant.

Q is FALSE when IN is FALSE und ET equal PT. Otherwise it is TRUE.

Thus, Q has a falling edge when the time indicated in PT in milliseconds has run out.

Graphic display of TOF behavior over time:



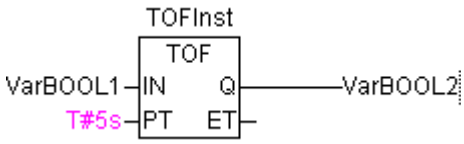
Declaration example:

```
TOFInst : TOF ;
```

Example in IL:

```
CAL     TOFInst(IN := VarBOOL1, PT := T#5s)
LD      TOFInst.Q
ST      VarBOOL2
```

Example in FBD:

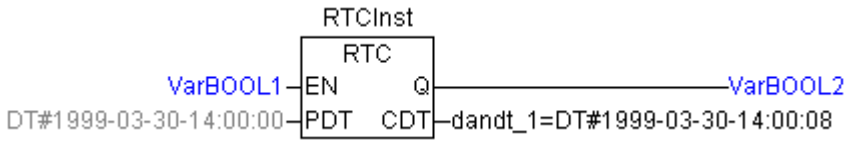


Example in ST:

```
TOFInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 :=TOFInst.Q;
```

RTC

The function block Runtime Clock returns, starting at a given time, the current date and time.



RTC(EN, PDT, Q, CDT) means:

EN and PDT are input variables type TIME. Q and CDT are output variables type BOOL respectively DATE_AND_TIME. When EN is FALSE, the output variables Q und CDT are FALSE respectively DT#1970-01-01-00:00:00.

As soon as EN becomes TRUE, the time of PDT is set, is counted up in seconds and returned in CDT as long as EN is TRUE (see example in the picture above). As soon as EN is reset to FALSE, CDT is reset to the initial value DT#1970-01-01-00:00:00. Please note that the time in PDT is only set by a rising edge.

Appendix E: The UTIL.LIB Library

This library contains an additional collection of various blocks which can be used for BCD conversion, bit/byte functions, mathematical help functions, as controller, signal generators, function manipulators and for analogue value processing.

As some of the functions and function blocks contain REAL variables, an accessory library named UTIL_NO_REAL exists in which these POUs are excluded.

BCD conversion

A byte in the BCD format contains integers between 0 and 99. Four bits are used for each decimal place. The ten decimal place is stored in the bits 4-7. Thus the BCD format is similar to the hexadecimal presentation, with the simple difference that only values between 0 and 99 can be stored in a BCD byte, whereas a hexadecimal byte reaches from 0 to FF.

An example: The integer 51 should be converted to BCD format. 5 in binary is 0101, 1 in binary is 0001, which makes the BCD byte 01010001, which corresponds to the value $51=81.

BCD_TO_INT

This function converts a byte in BCD format into an INT value:

The input value of the function is type BYTE and the output is type INT.

Where a byte should be converted which is not in the BCD format the output is -1.

Examples in ST:

```
i:=BCD_TO_INT(73);  (* Result is 49 *)
k:=BCD_TO_INT(151);         (* Result is 97 *)
l:=BCD_TO_INT(15);  (* Output -1, because it is not in BCD format *)
```

*INT_TO_BCD_*

This function converts an INTEGER value into a byte in BCD format:

The input value of the function is type INT, the output is type BYTE.

The number 255 will be outputted where an INTEGER value should be converted which cannot be converted into a BCD byte.

Examples in ST:

```
i:=INT_TO_BCD(49);  (* Result is 73 *)
k:=BCD_TO_INT(97);  (* Result is 151 *)
l:=BCD_TO_INT(100); (* Error! Output: 255 *)
```

Bit/byte functions

EXTRACT

Inputs to this function are a DWORD X, as well as a BYTE N. The output is a BOOL value, which contains the content of the N^{th} bit of the input X, whereby the function begins to count from the zero bit.

Examples in ST:

```
FLAG:=EXTRACT(X:=81, N:=4);     (* Result : TRUE, because 81 is binary
   1010001, so the 4th bit is 1 *)
FLAG:=EXTRACT(X:=33, N:=0);     (* Result : TRUE, because 33 is binary
   100001, so the bit '0' is 1 *)
```

PACK

This function is capable of delivering back eight input bits B0, B1, ..., B7 from type BOOL as a BYTE.

The function block UNPACK is closely related to this function.

PUTBIT

The input to this function consists of a DWORD X, a BYTE N and a BOOLean value B.

PUTBIT sets the N^{th} bit from X on the value B, whereby it starts counting from the zero bit.

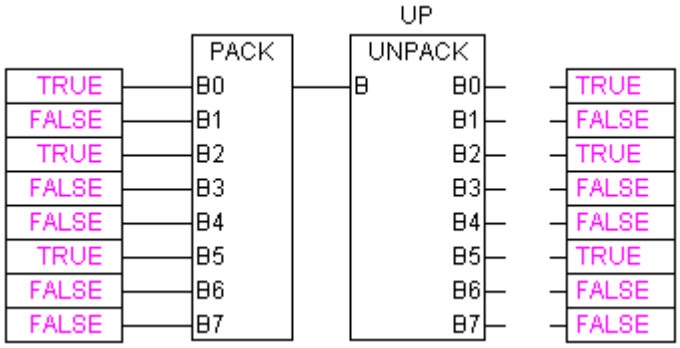
Example in ST:

```
A:=38;          (* binary 100110 *)
B:=PUTBIT(A,4,TRUE);        (* Result : 54 = 2#110110 *)
C:=PUTBIT(A,1,FALSE);       (* Result : 36 = 2#100100 *)
```

UNPACK

UNPACK converts the input B from type BYTE into 8 output variables B0,...,B7 of the type BOOL, and this is the opposite to PACK.

Example in FBD: Output:



Mathematical help functions

DERIVATIVE

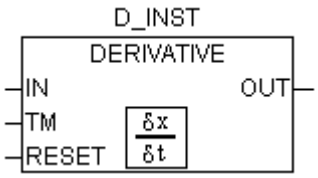
This function block approximately determines the local derivation.

The function value is delivered as a REAL variable by using IN. TM contains the time which has passed in msec in a DWORD and the input of RESET of the type BOOL allows the function block to start anew through the delivery of the value TRUE.

The output OUT is of the type REAL.

In order to obtain the best possible result, DERIVATIVE approximates using the last four values, in order to hold errors which are produced by inaccuracies in the input parameters as low as possible.

Block in FBD:



INTEGRAL

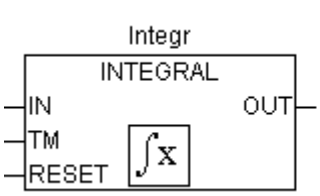
This function block approximately determines the integral of the function.

In an analogue fashion to DERIVATIVE, the function value is delivered as a REAL variable by using IN. TM contains the time which has passed in msec in a DWORD and the input of RESET of the type BOOL allows the function block to start anew with the value TRUE.

The output OUT is of the type REAL.

The integral is approximated by two step functions. The average of these is delivered as the approximated integral.

Block in FBD: Example: Integration of a linear function:



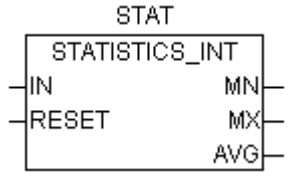
STATISTICS_INT

This function block calculates some standard statistical values:

The input IN is of the type INT. All values are initialised anew when the BOOLean input RESET is TRUE.

The output MN contains the minimum, MX of the maximum value from IN. AVG describes the average, that is the expected value of IN. All three outputs are of the type INT.

Block in FBD:



STATISTICS_REAL

This function block corresponds to STATISTICS_INT, except that the input IN is of the type REAL like the outputs MN, MX, AVG.

VARIANCE

VARIANCE calculates the variance of the entered values.

The input IN is of the type REAL, RESET is of the type BOOL and the output OUT is again of the type REAL.

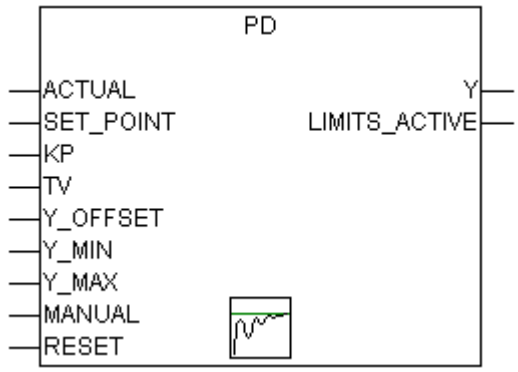
This block calculates the variance of the inputted values. VARIANCE can be reset with RESET=TRUE.

The standard deviation can easily be calculated as the square root of the VARIANCE.

Controllers

PD

The PD controller function block:



ACTUAL (actual value) and DESIRED (desired or nominal value) as well as KP, the proportionality coefficient, are all input values of the type REAL. TV is of the type DWORD and contains the derivative action time in msec. Y_OFFSET, Y_MIN and Y_MAX are of type REAL and are used for the transformation of the manipulated variable within a prescribed range. MANUAL, of type BOOL, switches to manual operation. RESET is of the type BOOL and serves to reset the controller.

$Y = KP \cdot (\Delta + TV\ \delta\Delta/\delta t) + Y\_OFFSET$ whereby Δ=SET_POINT-ACTUAL

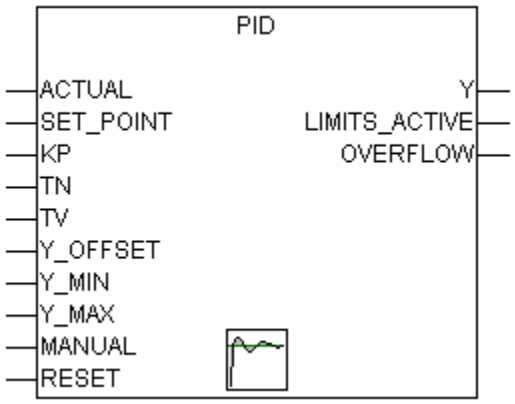
Y is also limited to the allowed range between Y_MIN and Y_MAX. If Y exceeds this range, LIMITS_ACTVE, a BOOLean output variable, becomes TRUE. If no limitation of the manipulated variable is desired, Y_MIN and Y_MAX are set to 0.

If MANUAL is TRUE, then the regulator is suspended , that is Y is not altered (by the controller), until MANUAL becomes FALSE, whereby the controller is re-initialized.

A P-controller is easily generated by setting TV to a fixed value of 0.

PID

The PID controller function block:



Unlike the PD controller, this function block contains a further DWORD input TN for the readjusting time in msec.

The output, the manipulated variable (Y) is again of type REAL, and contains, unlike the PD controller, an additional integral part:

$$Y = KP \cdot (\Delta + 1/TN \int \Delta(t)dt + TV\ \delta\Delta/\delta t) + Y\_OFFSET$$

The PID controller can be easily converted to a PI controller by setting TV=0.

Because of the additional integral part, an overflow can come about by incorrect parameterization of the controller, if the integral of the error Δ becomes too great. Therefore for the sake of safety a BOOLean output called OVERFLOW is present, which in this case would have the value TRUE. At the same time, the controller is suspended and will only be activated again by re-initialization.

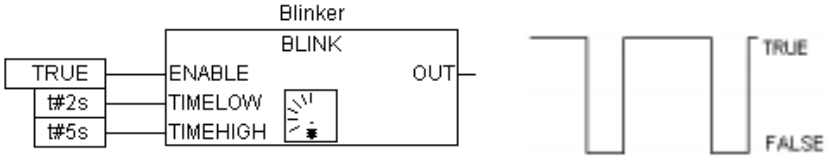
Signal generators

BLINK

The function block BLINK generates a pulsating signal. The input consists of ENABLE of the type BOOL, as well as TIMELOW and TIMEHIGH of the type TIME. The output OUT is of the type BOOL.

If ENABLE is set to TRUE, BLINK begins, to set the output for the time period TIMEHIGH to TRUE, and then afterwards to set it for the time period TIMELOW to FALSE.

Example in CFC:



GEN

The function generator generates typical periodic functions:

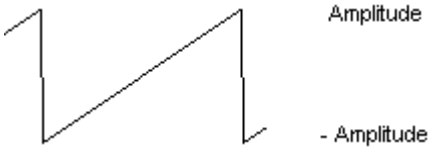
The inputs are a composition consisting of MODE from the pre-defined counting type GEN_MODE, BASE of the type BOOL, PERIOD of the type TIME, of two INT values CYCLES and AMPLITUDE and of the BOOLean RESET input.

The MODE describes the function which should be generated, whereby the enumeration values TRIANGLE and TRIANGLE_POS deliver two triangular functions, SAWTOOTH_RISE an ascending, SAWTOOTH_FALL a descending sawtooth, RECTANGLE a rectangular signal and SINE and COSINE the sine and cosine:

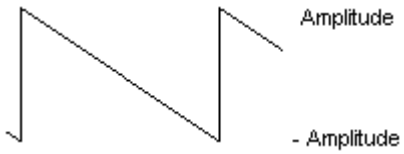
TRIANGLE:

TRIANGLE_POS:

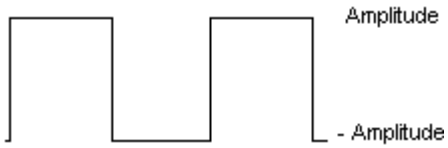
SAWTOOTH_RISE:



SAWTOOTH_FALL:



RECTANGLE:



SINUS:

COSINUS:

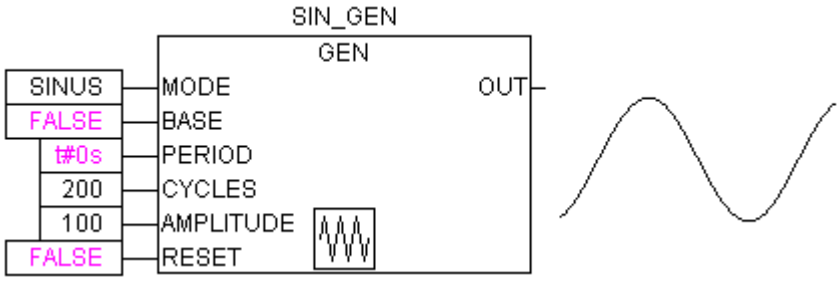
BASE defines whether the cycle period is really related to a defined time (BASE=TRUE) or whether it is related to a particular number of cycles, which means the number of calls of function block (BASE=FALSE).

PERIOD or CYCLES defines the corresponding cycle period.

AMPLITUDE defines, in a trivial way, the amplitude of the function to be generated.

The function generator is again set to 0 as soon as RESET=TRUE.

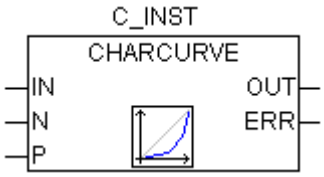
Example in FBD:



Function manipulators

CHARCURVE

This function block serves to represent values, piece by piece, on a linear function:



IN of the type INT is fed with the value to be manipulated. The BYTE N designates the number of points which defines the presentation function. This characteristic line is then generated in an ARRAY P[0..10] with P of the type POINT which is a structure based on two INT values (X and Y).

The output consists of OUT of the type INT, the manipulated value and BYTE ERR, which will indicate an error if necessary.

The points P[0]..P[N-1] in the ARRAY must be sorted according to their X values, otherwise ERR receives the value 1. If the input IN is not between P[0].X and P[N-1].X, ERR=2 and OUT contains the corresponding limiting value P[0]. Y or P[N-1].Y.

If N lies outside of the allowed values which are between 2 and 11, then ERR=4.

Example in ST:

First of all ARRAY P must be defined in the header:

```
VAR
    ...
    CHARACTERISTIC_LINE:CHARCURVE;
    KL:ARRAY[0..10] OF POINT:=(X:=0,Y:=0),(X:=250,Y:=50),
(X:=500,Y:=150),(X:=750,Y:=400),7((X:=1000,Y:=1000));
    COUNTER:INT;
    ...
END_VAR
```

Then we supply CHARCURVE with for example a constantly increasing value:

COUNTER:=COUNTER+10;

CHARACTERISTIC_LINE(IN:=COUNTER,N:=5,P:=KL);

The subsequent tracing illustrates the effect:

RAMP_INT

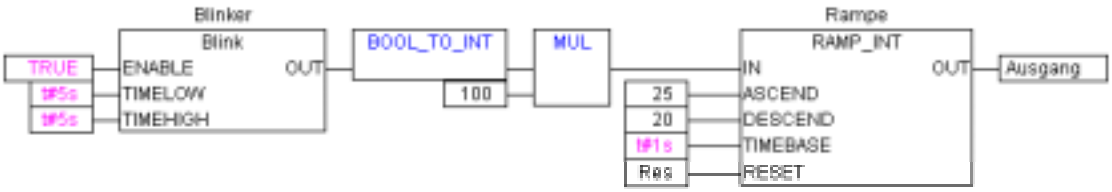
RAMP_INT serves to limit the ascendance or descendance of the function being fed:

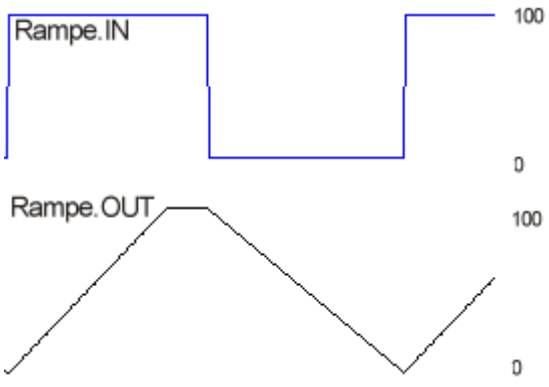
The input consists on the one hand out of three INT values: IN, the function input, and ASCEND and DESCEND, the maximum increase or decrease for a given time interval, which is defined by TIMEBASE of the type TIME. Setting RESET to TRUE causes RAMP_INT to be initialised anew.

The output OUT of the type INT contains the ascend and descend limited function value.

When TIMEBASE is set to t#0s, ASCEND and DESCEND are not related to the time interval, but remain the same.

Beispiel in CFC:





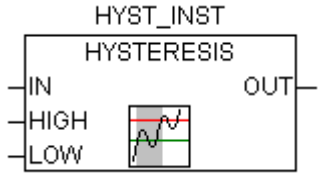
RAMP_REAL

RAMP_REAL functions in the same way as RAMP_INT, with the simple difference that the inputs IN, ASCEND, DESCEND and the output OUT are of the type REAL.

Analogue value processing

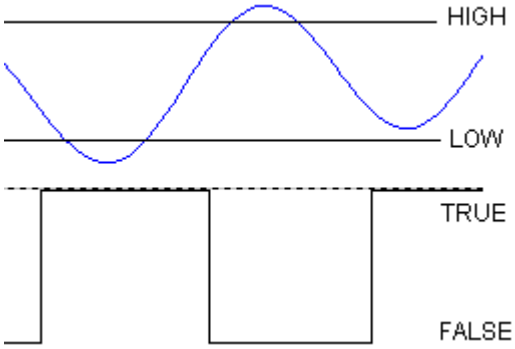
HYSTERESIS

The input to this function block consists of three INT values IN, HIGH and LOW. The output OUT is of the type BOOL.



If IN goes below the limiting value LOW, OUT becomes TRUE. If IN goes over the upper limit HIGH, FALSE is delivered.

An illustrative example:



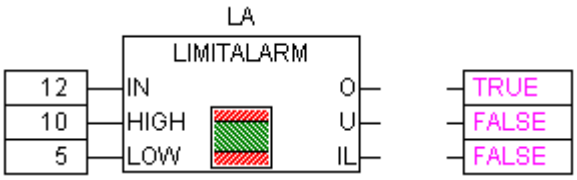
LIMITALARM

This function block specifies, whether the input value is within a set range and which limits it has violated if it has done so.

The input values IN, HIGH and LOW are each of the type INT, while the outputs O, U and IL are of the type BOOL.

If the upper limit HIGH is exceeded by IN, O becomes TRUE, and when IN is below LOW, U becomes TRUE. IL is TRUE if IN lies between LOW and HIGH.

Example in FBD: Result:



Appendix F: Operands in CoDeSys

Constants, variables, addresses and possibly function calls can appear as operands.

Constants in CoDeSys

BOOL Constants

BOOL constants are the logical values TRUE and FALSE.

TIME Constants

TIME constants can be declared in **CoDeSys**. These are generally used to operate the timer in the standard library. A TIME constant is always made up of an initial "t" or "T" (or "time" or "TIME" spelled out) and a number sign "#".

This is followed by the actual time declaration which can include days (identified by "d"), hours (identified by "h"), minutes (identified by "m"), seconds (identified by "s") and milliseconds (identified by "ms"). Please note that the time entries must be given in this order according to length (d before h before m before s before m before ms) but you are not required to include all time increments.

Examples of correct TIME constants in a ST assignment:

```
TIME1 := T#14ms;
TIME1 := T#100S12ms;        (*The highest component may be allowed to exceed its limit*)
TIME1 := t#12h34m15s;
```

the following would be incorrect:

```
TIME1 := t#5m68s;           (*limit exceeded in a lower component*)
TIME1 := 15ms;              (*T# is missing*)
TIME1 := t#4ms13d;          (*Incorrect order of entries*)
```

DATE Constants

These constants can be used to enter dates. A DATE constant is declared beginning with a "d", "D", "DATE" or "date" followed by "#". You can then enter any date with format Year-Month-Day.

Examples:

```
DATE#1996-05-06
d#1972-03-29
```

TIME_OF_DAY Constants

Use this type of constant to store times of the day. A TIME_OF_DAY declaration begins with "tod#", "TOD#", "TIME_OF_DAY#" or "time_of_day#" followed by a time with the format: Hour:Minute:Second. You can enter seconds as real numbers or you can enter fractions of a second.

Examples:
TIME_OF_DAY#15:36:30.123
tod#00:00:00

DATE_AND_TIME Constants

Date constants and the time of day can also be combined to form so-called DATE_AND_TIME constants. DATE_AND_TIME constants begin with "dt#", "DT#", "DATE_AND_TIME#" or "date_and_time#". Place a hyphen after the date followed by the time.

Examples:
DATE_AND_TIME#1996-05-06-15:36:30
dt#1972-03-29-00:00:00

Number Constants

Number values can appear as binary numbers, octal numbers, decimal numbers and hexadecimal numbers. If an integer value is not a decimal number, you must write its base followed by the number sign (#) in front of the integer constant. The values for the numbers 10-15 in hexadecimal numbers will be represented as always by the letters A-F.

You may include the underscore character within the number.

Examples:

14	(decimal number)
2#1001_0011	(dual number)
8#67	(octal number)
16#A	(hexadecimal number)

These number values can be from the variable types BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL or LREAL.

Implicit conversions from "larger" to "smaller" variable types are not permitted. This means that a DINT variable cannot simply be used as an INT variable. You must use the type conversion (see Chapter Appendix C: ,Type Conversions chapter in the appendix).

REAL/LREAL Constants

REAL and LREAL constants can be given as decimal fractions and represented exponentially. Use the standard American format with the decimal point to do this.

Example:
7.4 instead of 7,4
1.64e+009 instead of 1,64e+009

STRING Constants

A string is a sequence of characters. STRING constants are preceded and followed by single quotation marks. You may also enter blank spaces and special characters (umlauts for instance). They will be treated just like all other characters.

In character sequences, the combination of the dollar sign ($) followed by two hexadecimal numbers is interpreted as a hexadecimal representation of the eight bit

character code. In addition, the combination of two characters that begin with the dollar sign are interpreted as shown below when they appear in a character sequence:

$$	Dollar signs
$'	Single quotation mark
$L or $l	Line feed
$N or $n	New line
$P or $p	Page feed
$R or $r	Line break
$T or $t	Tab

Examples:

'w1Wüß?'
' Abby and Craig '
':-)'

Typed Literals

Basically, in using IEC constants, the smallest possible data type will be used. If another data type must be used, this can be achieved with the help of typed literals without the necessity of explicitly declaring the constants. For this, the constant will be provided with a prefix which determines the type.

This is written as follows: <Type>#<Literal>

<Type> specifies the desired data type; possible entries are: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, LREAL. The type must be written in uppercase letters.

<Literal> specifies the constant. The data entered must fit within the data type specified under <Type>.

Example:

var1:=DINT#34;

If the constant can not be converted to the target type without data loss, an error message is issued:

Typed literals can be used wherever normal constants can be used.

Variables

Variables can be declared either locally in the declaration part of a POU or in a global variable list.

The variable identifier may not contain any blank spaces or special characters, may not be declared more than once and cannot be the same as any of the keywords. Capitalization is not recognized which means that VAR1, Var1, and var1 are all the same variable. The underscore character is recognized in identifiers (e.g., "A_BCD" and "AB_CD" are considered two different identifiers). An identifier may not have more than one underscore character in a row. The length of the identifier, as well as the meaningful part of it, are unlimited.

Variables can be used anywhere the declared type allows for them.

You can access available variables through the Input Assistant.

System Flags

System flags are implicitly declared variables that are different on each specific PLC. To find out which system flags are available in your system, use the command '**Insert**' '**Operand**' An Input Assistant dialog box pops up, select the category **System Variable**.

Accessing variables for arrays, structures and POUs.

Two-dimensional array components can be accessed using the following syntax:
<Fieldname>[Index1, Index2]

Structure variables can be accessed using the following syntax:
<Structurename>.<Variablenname>

Function block and program variables can be accessed using the following syntax:
<Functionblockname>.<Variablename>

Addressing bits in variables

In integer variables, individual bits can be accessed. For this, the index of the bit to be addressed is appended to the variable, separated by a dot. The bit-index can be given by any constant. Indexing is 0-based. Example:

```
a : INT;
b : BOOL;
...
a.2 := b;
```

The third bit of the variable a will be set to the value of the variable b.

If the index is greater than the bit width of the variable, the following error message is issued: Index '<n>' outside the valid range for variable '<var>'!

Bit addressing is possible with the following variable types: SINT, INT, DINT, USINT, UINT, UDINT, BYTE, WORD, DWORD.

If the variable type does not allow it, the following error message is issued: „Invalid data type '<type>' for direct indexing“

A bit access must not be assigned to a VAR_IN_OUT variable!

Addresses

Address

The direct display of individual memory locations is done through the use of special character sequences. These sequences are a concatenation of the percent sign "%", a range prefix, a prefix for the size and one or more natural numbers separated by blank spaces.

The following range prefixes are supported:

I	Input
Q	Output
M	Memory location

The following size prefixes are supported:

X	Single bit
None	Single bit
B	Byte (8 Bits)
W	Word (16 Bits)
D	Double word (32 Bits)

Examples:

%QX7.5 and %Q7.5	Output bit 7.5
%IW215	Input word 215
%QB7	Output byte 7
%MD48	Double word in memory position 48 in the memory location.
%IW2.5.7.1	depending on the PLC Configuration

The current PLC Configuration for the program determines whether or not an address is valid.

Note: Boolean values will be allocated bytewise, if no explicit single-bit address is specified. Example: A change in the value of varbool1 AT %QW0 affects the range from QX0.0 to QX0.7.

Memory location

You can use any supported size to access the memory location.

For example, the address %MD48 would address bytes numbers 192, 193, 194, and 195 in the memory location area (48 * 4 = 192). The number of the first byte is 0.

You can access words, bytes and even bits in the same way: the address %MX5.0 allows you to access the first bit in the fifth word (Bits are generally saved wordwise).

Functions

In ST a function call can also appear as an operand.

Example:

Result := Fct(7) + 3;

TIME()-Funktion

This function returns the time (based on milliseconds) which has been passed since the system was started.

The data type is TIME.

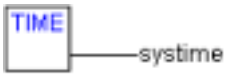
Example in IL:

```
TIME
ST      systime      (* Result e.g.: T#35m11s342ms *)
```

Example in ST:

```
systime:=TIME();
```

Example in FUP:



Appendix G: Command Line/Command File Commands

Command Line Commands

When CoDeSys is started, you can add commands in the command line which will be asserted during execution of the program. These commands start with a „/“. Capitalization/Use of small letters is not regarded. The commands will be executed sequentially from the left to the right.

/online	Immediately after start CoDeSys tries to go online with the current project.
/run	After login CoDeSys starts the application program. Only valid in combination with /online.
/show ...	Settings for the CoDeSys frame window can be made.
/show hide	The window will not be displayed, it also will not be represented in the task menu.
/show icon	The window will be minimized in display.
/show max	The window will be maximized in display.
/show normal	The window will be displayed in the same status as it was during the last closing.
/out <outfile>	All messages are displayed in the message window and additionally are written in the file <outfile>.
/cmd <cmdfile>	After starting the commands of the <cmdfile> get executed.

Example for a command line:

The project ampel.pro gets opened, but no window opens. The commands included in the command file command.cmd will be executed.

C:\ampel.pro /show hide /cmd command.cmd

Command File (cmdfile) Commands

See the following table for a list of commands, which can be used in a command file (<cmdfile>). The command file you can call by a command line (see above) aufrufen können. Capitalizing/Use of small letters is not regarded. The command line will be displayed as a message in the message window and can be given out in a message file (see below). Additionally to the command a „@“ is prefixed. All signs after a semicolon (;) will be ignored (comment). Keywords can be used in the

command parameters. A list of the keywords you find subsequent to the following tables of command descriptions.

Commands for controlling any subsequent commands:

onerror continue	The subsequent commands will be executed even if an error occurs.
onerror break	The subsequent commands will not be executed any more if an error has been detected.

Commands of the online menu:

online login	Login with the loaded project ('Online Login')
online logout	Logout ('Online' 'Logout')
online run	Start of the application program ('Online' 'Run')
online sim	Switch on of simulation mode (✓) 'Online' 'Simulation')
online sim off	Switch off of simulation mode ('Online' 'Simulation')

Commands of the file menu:

file new	A new project is created ('File' 'New')
file open <projectfile>	The project <projectfile> will be loaded ('File' 'Open')
file close	The current project will be closed ('File' 'Close')
file save	The current project will be stored ('File' 'Save')
file saveas <projectfile>	The current project will be saved with the file name <projectfile> ('File' 'Save as')
file quit	CoDeSys will be closed ('File' 'Exit')

Commands of the project menu:

project compile or project build	The project that is loaded will be incrementally compiled ('Project' 'Build')
project rebuild	The project that is loaded will be compiled in full ('Project' 'Rebuild')
project clean	Compilation information and Download-Information in the current project will be deleted ('Project' 'Clean Project')
project compile	The current project will be compiled by "Rebuild all" ('Project' 'Rebuild all')
project import <file1> ... <fileN>	The files <file1> ... <fileN> get imported into the current project ('Project' 'Import')
project export <expfile>	The current project will be exported in the file <expfile> ('Project' 'Export')

project expmul	Each object of the current project will be exported in an own file, which gets the name of the object.

Commands for the control of the message file:

out open <msgfile>	The file <msgfile> opens as message file. New messages will be appended
out close	The currently shown message file will be closed.
out clear	All messages of the currently opened message file will be deleted.

Commands for the control of messages:

echo on	The command lines will be displayed as messages.
echo off	The command lines will not be displayed as messages.
echo <text>	<text> will be displayed in the message window.

Commands for the control of replace of objects respectively for the control of files for import, export, copy:

replace yesall	Replace all (any 'query on' command will be ignored; no dialogs will open)
replace noall	Replace none (any 'query on' command will be ignored; no dialogs will open)
replace query	If a 'query on' command is set, then a dialog will open regarding the replacing of the objects even if there is a 'replace yesall' or 'replace noall' command

Commands for the control of the default parameters of CoDeSys dialogs:

query on	Dialogs are displayed and need user input
query off ok	All dialogs respond as if the user had clicked on the 'OK' button
query off no	All dialogs respond as if the user had clicked on the 'No' button
query off cancel	All dialogs respond as if the user had clicked on the 'Cancel' button

Command for calling command files as subprograms:

call <parameter1> ... <parameter10>	Command files will be called as subprograms. Up to 10 parameters may be passed. In the file that is called, the parameters can be accessed with $0 - $9.
call <parameter1> ... <parameter10>	Command files are called as subroutines. Up to ten parameters can be consigned. In the subroutine called you can access the parameters using $0 - $9.

Setting of directories used by CoDeSys:

dir lib <libdir>	Sets <libdir> as the library directory
dir compile <compiledir>	Sets <compiledir> as the directory for the compilation files

Delaying processing of the CMDFILE:

delay 5000	Waits 5 seconds

Controlling the Watch and Receipt Manager:

watchlist load <file>	Loads the Watchlist saved as <file> and opens the corresponding window ('Extras' 'Load Watchlist')
watchlist save <file>	Saves the current Watchlist as <file> ('Extras' 'Save Watchlist')
watchlist set <text>	Gives a previously loaded Watchlist the name <text> ('Extras' 'Rename Watchlist')
watchlist read	Updates the values of the Watch variables ('Extras' 'Read receipt')
watchlist write	Fills the Watch variables with the values found in the Watchlist ('Extras' 'Write receipt')

Linking libraries:

library add <library file1> <library file2> .. <library fileN>	Attaches the specified library file to the library list of the currently open project. If the file path is a relative path, the library directory entered in the project is used as the root of the path.
library delete [<library1> <library2> .. <libraryN>]	Deletes the specified library, or (if no library name is specified) all libraries from the library list of the currently open project.

Copying objects:

object copy <source project file> <source path> <target path>	Copies objects from the specified path of the source project file to the target path of the already opened project. If the source path is the name of an object, this will be copied. If it is a folder, all objects below this folder will be copied. In this case, the folder structure below the source folder will be duplicated. If the target path does not yet exist, it will be created.

Entering communications parameters (gateway, device):

gateway local	Sets the gateway on the local computer as the current gateway.
gateway tcpip <Address> <Port>	Sets the gateway in the specified remote computer as the current gateway.

	<Address>: TCP/IP address or hostname of the remote computer <Port>: TCP/IP port of the remote gateway Important: Only gateways that have no password set can be reached!
device guid <guid>	Sets the device with the specified GUID as the current device. GUID must have the following format: {01234567-0123-0123-0123-0123456789ABC} The curly brackets and the hyphens must appear at the specified positions.
device instance <Instance name>	Sets the instance name for the current device to the name specified
device parameter <Id> <Value>	Assigns the specified value, which will then be interpreted by the device, to the parameter with the specified ID.

System call:

system <command>	Carries out the specified operating system command.

Select target system

target <Id>	Sets the target platform for the current project.

Commands concerning managing the project in the ENI project data base::

Please regard: The following placeholders are used in the subsequent description of the particular commands:

<category>: Replace by "project" or "shared" or "compile" depending on which of the following data base categories is concerned: Project Objects, Shared Objects, Compile Files

<POUname>: Name of the object, corresponds to the object name which is used in CoDeSys.

<Objecttype>: Replace by the shortcut, which is appended as an extension to the POU name of the object in the data base, and which reflects the object type (defined by the list of object types, see ENI Administration, 'Object Types').

Example: Object "GLOBAL_1.GVL" -> the POU name is "GLOBAL_1", the object type is "GVL" (global variables list)

<comment>: Replace by a comment text (embraced by single quotation marks), which will be stored in the version history with the particular action.

Commands to configurate the project data base link via the ENI Server:

eni on The option 'Use source control (ENI)' will be
eni off activated resp. deactivated

	(Dialog 'Project' 'Options' 'Project source control')
eni project readonly on **eni project readonly off**	The option 'Read only' for the data base category 'Project objects' will be activated resp. deactivated (Dialog 'Project' 'Options' 'Project objects')
eni shared readonly on **eni shared readonly off**	The option 'Read only' for the data base category 'Shard objects' will be activated resp. deactivated (Dialog 'Project' 'Options' 'Shared objects')
eni set local <POUname>	The object will be assigned to category 'Local', i.e. it will not be stored in the project data base (Dialog 'Project' 'Object' 'Properties' 'Data base-connection')
eni set shared <POUname>	The object will be assigned to category 'Shared objects' (Dialog 'Project' 'Object' 'Properties' 'Data base-connection')
eni set project <POUname>	The object will be assigned to category 'Project objects' (Dialog 'Project' 'Object' 'Properties' 'Data base-connection')
eni <category> server <TCP/IP_Address> <Port> <Projectname> <Username> <Password>	Configures the connection to the ENI Server for the category 'Project objects' (Dialog 'Project' 'Options' 'Project data base'); Example: `eni project server localhost 80 batchtest\project EniBatch Batch` -> TCP/IP-Address = localhost, Port = 80, Project name = batchtest\project, User name = EniBatch, Password = Batch
eni compile sym on **eni compile sym off**	The option 'Create ASCII symbol information (.sym)' for the objects of category 'Compile files' will be activated/deactivated (Dialog 'Project' 'Options' 'Project source control' 'ENI settings' for 'Compile files')
eni compile sdb on **eni compile sdb off**	The option 'Create binary symbol information (.sym)' for the objects of category 'Compile files' will be activated/deactivated (Dialog 'Project' 'Options' 'Project source control' 'ENI settings' for 'Compile files')
eni compile prg on **eni compile prg off**	The option 'Create boot project' for the objects of category 'Compile files' will be activated/deactivated (Dialog 'Project' 'Options' 'Project source control' 'ENI settings' for 'Compile files')

Commands of the menu 'Project' 'Data Base Link' for working with the data base:

eni set <category>	The object gets assigned to the named data base

	category ('Define')
eni set <category>set <Objecttype>:<POUname> <Objecttype>:<POUname>	The objects which are listed separated by spaces will be assigned to the named data base category. ('Multiple Define') Example: "eni set project pou:as_fub pou:st_prg" -> the objects (pou) as_fub and st_prg get assigned to category 'Project objects'
eni <category> getall	The latest version of all objects of the named category will be called from the data base ('Get All Latest Versions')
eni <category>get <Objecttype>:<POUname> <Objecttype>:<POUname>	The objects of the named category, which are listed separated by spaces will be called from the data base. ('Multiple Define'). ('Abrufen') Example: "eni project get pou:as_fub gvl:global_1" -> the POU as_fub.pou and the global variables list global_1.gvl will be called from the data base
eni <category> checkoutall "<comment>"	All objects of the named category will be checked out from the data base. The defined comment will be stored with the check-out-action in the version history.
eni <category> checkout "<comment>" <Objecttype>:<POUname> <Objecttype>:<POUname>	All objects (Objecttype:POUname) which are listed separated by spaces will be checked out from the data base. The defined comment will be stored with the check-out-action in the version history for each particular object. Example: "eni project checkout "for working on xy" pou:as_fub gvl:global_1" -> The POU as_fub and the global variables list global_1 will be checked out and the comment "for working on xy" will be stored with this action
eni <category>checkinall "<comment>"	All objects of the project, which are under source control in the project data base, will be checked in. The defined comment will be stored with the check-in-action.
eni <category> checkin "<comment>" <Objecttype>:<POUname> <Objecttype>:<POUname>	All objects (Objecttype:POUname) which are listed separated by spaces will be checked in to the data base. The defined comment will be stored with the check-in-action in the version history for each particular object. (see above: check out) The defined comment will be stored with the check-in-action in the version history for each particular

object.

Keywords for the command parameters:

The following keywords, enclosed in "$", can be used in command parameters:

$PROJECT_NAME$	Name of the current CoDeSys project (file name without extension ".pro", e.g. "project_2")
$PROJECT_PATH$	Path of the directory, where the current CoDeSys project file is (without indication of the drive and without a backslash at the end, e.g. "projects\sub1").
$PROJECT_DRIVE$	Drive, where the current CoDeSys project is (without backslash at the end, e.g. "D:")
$COMPILE_DIR$	Compile directory of the current CoDeSys project (with indication of the drive and without backslash at the end, e.g. "D:\codesys\compile")
EXE_DIR	Directory where the codesys.exe file is (with indication of the drive and without backslash at the end, e.g. D:\codesys)

Example of a command file:

A command file like shown below will open the project file ampel.pro, will then load a watch list, which was stored as w.wtc, will then start the application program and write – after 1 second delay - the values of the variables into the watch list watch.wtc (which will be saved) and will finally close the project.

```
file open C:\work\projects\CoDeSys_test\ampel.pro
query off ok
watchlist load c:\work\w.wtc
online login
online run
delay 1000
watchlist read
watchlist save c:\work\watch.wtc
online logout
file close
```

A command file is called in a command line like shown here:

"<Path of CoDeSys-Exe>" /cmd "<Path of cmd file>"

Appendix H: Siemens Import

In the **"Project" "Siemens Import"** submenu, you will find commands which allow you to import POUs and variables from Siemens STEP5 files. The command "Import from a SEQ symbol file" allows you to import global variables from STEP5 symbol files. Run this command before either the command **"Import from a S5 project file"** so that readable symbol names can be created when the POUs are imported. These two commands allow you to import POUs from STEP5 program files. When this done, the POUs are inserted into the open CoDeSys project. You can select whether the POUs will remain in the STEP5 IL language or be converted to an IEC language.

We recommend that the CoDeSys project into which you are importing be empty. Of course, you must be certain that the library standard.lib is linked to your project, otherwise you will be unable to import the counter and the timer.

Import from a SEQ Symbol File

SEQ format is a common format for symbol files in a STEP5 project. Symbol assignments can be read from SEQ symbol files (*.seq). A symbol assignment contains an absolute address for a S5 program element (input, output, memory location, etc.), a corresponding symbol identifier and may also contain comments about the symbol. A SEQ file is text file that contains one assignment of this type per line. Each of the "Fields" in the line are separated by Tabs. Also each line can only hold one comment which must begin with a semicolon.

The symbol assignments in the SEQ file will be translated into global variable declarations based on IEC 61131-3. The symbolic name, the address and the comment (if available) will be transferred during this process. The address will be adapted to IEC 61131-3 (Percent sign, etc.). Since a S5 symbol name can contain characters that are not permitted in an IEC identifier, the names will be changed if necessary. Invalid characters will be replaced by the underscore character. Should there be more than one underscore character in a row, every second one would be replaced by a valid character (e.g., "0"). If a symbol name is changed during the conversion, the original name will be added in a comment after the change. SEQ comment lines will be transferred as comments. Multiple blocks of global variables can be created. Each block consists of less than 64K of text.

The SEQ format described is used in Siemens STEP5-PG Software and in ACCON-PG from DELTALOGIC.

You first select the SEQ file in a standard Windows dialog box. Then perform the import, when this is done the global variable list will be compiled. Errors may arise during this process when STEP5 identifiers are converted into IEC61131-3 compatible identifiers. For example, both STEP5 identifiers "A!" and "A?" would be converted into the IEC identifier "A_". At this point the following message would appear, "Multiple declarations with the same identifier A_". Change one of the variables.

Under absolutely no other circumstances should you make any changes to the global variable list. If you identify addresses that are valid in a Siemens PLC but are invalid in your Controller: Leave them alone for now, even if you get a thousand error

messages while compiling. The addresses are needed exactly as they are in order to import the POUs.

If the project into which you are importing already contains a declaration for a global variable x with its address (e.g., "%MX4.0"), you may find that the SEQ import contains a variable defined with the same address. This is allowed in IEC 61131-3 but is generally not in the best interest of the user. No error message will appear, but your program may not function as it should since the address will be used in different POUs with no contextual reference. To avoid this problem, it is best to import into an empty project or into a project in which no absolute addresses have been used up to this point.

STEP5 Program Organization Units can be imported, once the SEQ import has been performed. You can also add the inputs and outputs that will be used in the PLC Configuration. These are not required for the STEP5 import but the addresses will be checked and may show up as errors when you rebuild the project.

Import from a S5 Project File

POUs can read from Siemens S5 program files (*.s5d). The code that it uses is MC5 Code that can be run by S5 SPS. In general, MC5 Code corresponds with the STEP5 Instruction List (without symbol names) with which the programmer is familiar. The S5D also contains the line comments from the STEP5 Instruction List. Since an S5D file contains only absolute addresses with no symbol names, CoDeSys searches for the symbol names among the current CoDeSys project variables. If none are found, the absolute address is left unchanged. Therefore, if you feel the symbol name is useful, import the SEQ file before the S5 file.

You first select the S5D file in a standard Windows dialog box. Another box pops up which contains the list of POUs from which you can select. It is best to select all of them. You can also select to leave the POUs in the STEP5 IL language or to convert them to IL, LD or FBD.

Symbol names will be used in place of absolute names as much as possible. If CoDeSys finds the instruction "U M12.0" during the import, it will search for a global variable set at memory location M12.0. The first declaration that fits this description will be taken and the instruction will be imported as "U-Name" instead of "U M12.0" (the name of the identifier for the memory location is M12.0).

At times additional variables may be needed during an import or code conversion. These additional variables will be declared globally. For example, R_TRIG instances are needed to reproduce edge-triggered inputs (e.g., in a S5 counter).

Converting S5 to IEC 61131-3

If you select an IEC language as your target language for a STEP5 import, you must be aware that portions of your project cannot be converted into IEC 61131-3. If part of a S5 POU contains code that cannot be converted into IEC 61131-3, an error message will be generated and the critical portion of the original STEP5 IL code will be inserted as a comment in the IEC POU. You must then replace or rewrite this code. System commands that only function in a specific S5 CPU cannot be converted into IEC. The "STEP5 Core Command List" can be converted into IEC code with a click of a button despite the fact that STEP5 is enormously different in the way it is conceived.

The core command list that can be converted to IEC 61131-3 contains all commands that can be converted to LD or FBD in a STEP5 programming system and also all commands that are allowed in a STEP5-PB (Program Block). In addition, of the STEP5 commands allowed only in IL or in FB's (function blocks), the commands that can be converted to IEC are primarily those that are available in every S5 CPU (e.g., absolute and conditional jumps, shift commands, etc.)

The only exception or limitation for conversion is related to resetting timers which can be done in STEP5 but not normally in IEC 61131-3.

The individual convertible commands:

- U, UN, O, ON, S, R, = with the following bit operands: I (inputs), O (outputs), M (memory locations), S (S memory locations), D (Data in data blocks)
- U, UN, O, ON with the following operands: T (Timer), C (Counter)
- S, R with the following operands: C
- SU, RU, P, PN with the following operands: E, A, M, D
- O, O(, U(,)
- L, T with the following operand ranges: E, A, M, D, T, C, P (Periphery) and operand sizes: B (byte), W (word), D (double word), L (left byte), R (right byte)
- L with the following constant formats: DH, KB, KF, KH, KM, KT, KZ, KY, KG, KC
- SI, SE, SA with the following operands: T
- ZV, ZR with the following operands: C
- +, -, X, : with the following operands: F (fixed point number), G (floating point number)
- +, - with the following operands: D (32 bit fixed point number)
- !=, ><, >, <, >=, <= with the following operands: F, D, G
- ADD with the following operands: BF, KF, DH
- SPA, SPB with the following operands: PB, FB (with most parameter types), SB
- A, AX with the following operands: DB, DX
- BE, BEA, BEB
- BLD, NOP, ***
- UW, OW, XOW
- KEW, KZW, KZD
- SLW, SRW, SLD, RRD, RLD
- SPA=, SPB=
- SPZ=, SPN=, SPP=, SPM=
- TAK
- D, I
- Most of the formal operand commands

Unconvertible Commands

- U, UN, O, ON, S, R, = with the following bit operands: Timer and counter bits (T0.0, C0.0)
- L, T with the following operand ranges: Q (expanded periphery)
- LC with the following operands: T, C
- SV, SS, R, FR with the following operands: T
- FR with the following operands: C
- Formal operand commands for starting, resetting and releasing timers
- All commands with operands from the ranges BA, BB, BS, BT (operating system data).

SPA, SPB with the following operands: OB (works only with certain S5's and certain OBs)
BA, BAB with the following operands: FX
E, EX with the following operands: DB, DX
STP, STS, STW
DEF, DED, DUF, DUD
SVW, SVD
SPO=, SPS=, SPR
AS, AF, AFS, AFF, BAS, BAF
ENT
SES, SEF
B with the following operands: DW, MW, BS
LIR, TIR, LDI, TDI, TNW, TXB, TXW
MAS, MAB, MSA, MSB, MBA, MBS
MBR, ABR
LRW, LRD, TRW, TRD
TSG
LB, TB, LW, TW with the following operands: GB, GW, GD, CB, CW, CD
ACR, TSC
BI
SIM, LIM

If you examine the commands that cannot be converted you will see that they are generally special commands that are only available on certain CPUs. The standard commands that cannot be converted to IEC are: loading BCD coded timer or counter values (LC T, LC C), timer types SV and SS, and resetting timers.

Data Blocks

STEP5 data blocks are converted into POUs (Program Organization Units) that have a header but no code. This is convenient if the data blocks are used as normal variable ranges but inconvenient if attempts have been made to manually implement concepts like instance data blocks in the STEP5 program.

Other Problems when Importing from STEP5

The STEP5 import can be improved manually in the following ways.

1. Time values in word variables

In STEP5 a time value is allowed in every word address be it in the memory location area or in a data block. This is not allowed in IEC 61131-3, TIME variables or constants are not compatible with WORD addresses. This can result in the creation of erroneous command sequences when importing from STEP5. This will not happen if you open a data block and select the time format (KT) for the address in question. In other words, this error only occurs when the STEP5 program is worth the effort of improving it. When it does occur, you will see the message "Incompatible Types: Cannot convert WORD to TIME." or "Incompatible Types: Cannot convert TIME to WORD." You must then modify the declaration for the WORD variable (if available) and turn it into a TIME variable.

2. Failure to Access Data Blocks

There are no data blocks in IEC 61131-3 and it is impossible completely to recreate them in IEC. In STEP5 they are used as normal variable ranges (almost like a

memory location ranges), and also in the form of arrays (B DW), pointers (B MW100 A DB 0) or unions (byte, word or double word access in DBs). STEP5 conversion can only convert DB access if it is somewhat structured. When attempting to access DBs you must know which DB is open (A DB). You must be aware of this when the A DB operation is closer to the beginning in the same POU or when the DB number is included with the POU as a formal parameter. If A DB is not found in front of the first DB access, the POU cannot be converted. The warning "No open data block (insert an A DB)" notifies you that this is the case. In the converted POU, you will see access to an undefined variable named "ErrorDW0" (for example) that will cause an error message to be generated when the newly converted POU is compiled. You can then replace the variables with access to the correct DB (e.g., replace "ErrorDW0" with "DB10.DW0"). The other option is to discard the converted POU and insert an A DB at the beginning of the POU in STEP5.

A STEP5 POU that accesses data words (data bytes, etc.) should always open the data block first. If necessary, the POU should be improved before being imported by inserting the appropriate A DB command preferably at the beginning of the POU. Otherwise the converted POU will have to be edited after the fact.

If there is more than one A BD operation that must be partially skipped, the conversion may have a errors, i.e., code may be generated that accesses the wrong DB.

3. Higher Concepts Related to Data Block Access

In STEP5 you have the option of creating something similar to instances by having the Code block open an indexed version of a data block. This could be done with the following sample code sequence:

```
L  KF   +5
T  MW   44
B  MW   44
A  DB   0
```

The DB5 is opened at the end of this sequence (in general, the DB whose number is found in the memory location word %MW44 will be opened). This type of access is not recognized in the conversion which means that the following changes have to be made after the conversion:

First, all DBs must be imported that act as instance DBs , e.g., DB5 and DB6. They will be imported as normal IL, LD or FBD POUs whichever you prefer. The POUs do not have a code, but rather a header with definitions of local variables. Type instances can now be created from these POUs. Create a user-defined type (e.g., named DBType) and insert the local variables and converted DBs as components. Then create global instances of this type by writing to a global variable list:

```
VAR_GLOBAL
   DB5, DB6 : DBType;
END_VAR
```

You can now delete the converted DBs from your project.

Then you have to create a copy of the indexed version of the DBs by giving the corresponding POU another VAR_INPUT parameter of the type DBType. Data access within the POU must now be redirected to this instance. You must then include one of the instance DBs as an actual parameter when you open it.

4. The so-called integrated S5 function blocks that have a STEP5 access interface have a special function but their implementation is either not written in STEP5 (or MC5) or is protected by a special mechanism. POUs of this kind are generally firmware and can only be "imported as and interface". The implementation portion of this type of POU is empty. These POUs must generally be reprogramed after being converted.

5. There are also firmware OBs that have no interface but whose code is in 805xx Assembler (as an example) and not in STEP5. This mainly affects the PID regulator listed as OB251 which obtains its parameters and local variables through a separate (data) block that you can select. Neither the PID regulator, the corresponding data block or other POUs that use regulators to access the data block can be converted to IEC. The IEC code that is created for data blocks and other POUs during the conversion is meaningless without the PID regulator. The meaning of the individual program parts can be found in the programming handbook for the CPU.

6. Configuration data blocks (like DB1 [S5-95U], DX0, and DX2) are sometimes used to configure S5 CPUs and other assemblies that were converted into useless IEC POUs. The meaning of much of this type of data can be found in the programming handbook for the CPU. For the rest you must use a S5 programming system that can evaluate the configuration DBs . The configuration affects settings for communication, analog value processing, multiprocessing, etc. Therefore, it is useless to even think about working with these POUs on a non-Siemens SPS.

Once the import is complete, you have to find the errors that are shown and then fix, add to and rewrite the affected spots. These spots are marked with comments like:

(*Warning! Unconvertible STEP5 code shown as comment:*)

This is followed by the unconvertible code which is also shown as a comment.

Finally, you must check the addresses. Original Siemens addresses are created during the import. These addresses have the following format:

Bits: Byte-Offset.Bit-Nummer

Non-Bits:Byte-Offset

Also word addresses that follow each other in sequence will overlap (simply due to the fact that the numbers in the addresses are byte offsets). This means that %MW32 and %MW33 have an overlapping byte which is %MB33 (only on a Siemens SPS). On your SPS, %MW32 and %MW33 would not normally have anything to do with each other.

Your PLC may have more hierarchies. For example, non-bits have several interlocking levels ("%MW10.0.0" as WORD). You can either make changes to the addresses to make them compatible with your PLC or you can try to leave them out entirely. Proceed very cautiously! In the original Siemens program, it is quite common that word access and bit or byte access is made in the same memory location. When imported into CoDeSys, accesses of this type will only be correctly compiled for data blocks. In this case, CoDeSys creates WORD variables for the words in the DBs . Then when WORD accesses word x in DB y there are no problems. Attempts to access the left or right byte in word x, a double word or a bit will then be compiled into more complex expressions. This cannot be done with memory locations, inputs or outputs since this can't be done with a standard access method (e.g., word access). If you are working with %MX33.3 and with %MB33 or

%MW32 or %MD30 you must go to the effort of converting them manually. The IEC program generated by the CoDeSys import will definitely not work correctly. Open a cross reference list containing all inputs, outputs and memory locations to find out which accesses are important. Remove the mixed accessed manually.

Target Settings

The Target Settings are located as object in the resources' tab. This is where it is determined on which controller (target) and with which settings the project shall run. After the command 'Project' 'New' the **Target Settings** " dialogue opens automatically and one is requested to select one of the predefined target configurations offered under **Configuration**.

The selection list depends upon the Target Support Packages (TSP) installed in the computer. These describe platform-specific basic configurations and simultaneously ascertain to what extent these can be further adjusted by the user in the dialogues of the target-system settings. If no Target Support Package is installed, the only choice is the setting '**None**', which automatically leads into the simulation modeIf a target, for which there is no valid licence in the computer, is selected from a TSP, another target is requested for selection.

As long as the set configuration is not generally protected with 'HideSettings' (in which case only the name of the configuration is shown), four tabs are available for final adjustment or for presentation of the target-system settings:

- **Target Platform**
- **Memory Layout**
- **General**
- **Network Functionality**

Attention: Please regard, that each modification of the settings which are predefined can lead to a severe change of performance and behaviour of the target system !

Press button Default to set back the target settings to the standard configuration, which is pre-defined by the target file

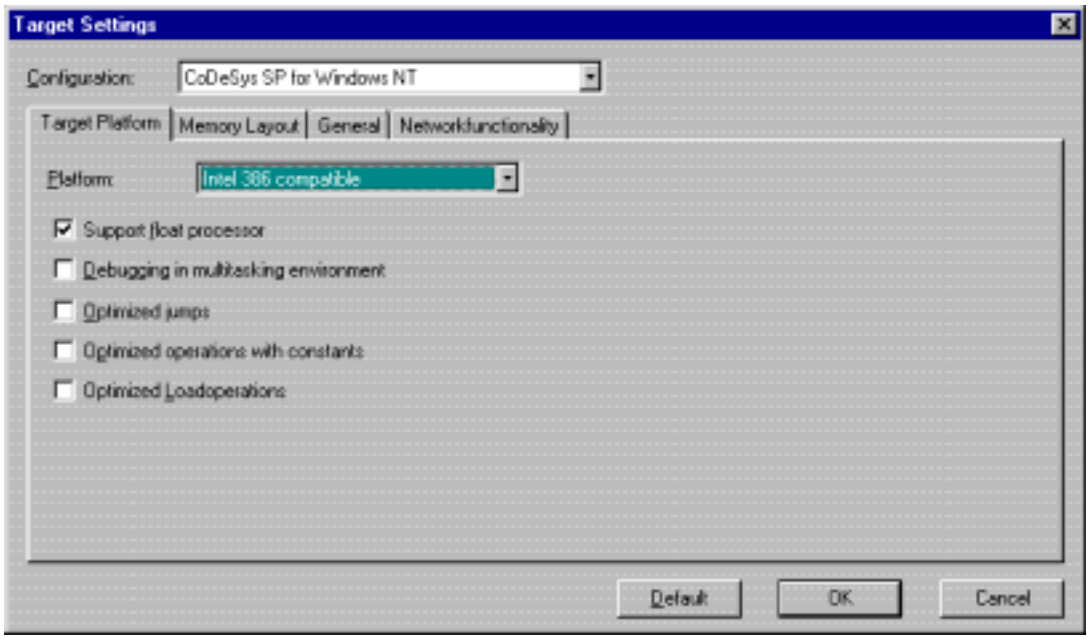
The description of the possible dialog items of the categories Memory Layout, General and Networkfunctionality is valid for all platforms. Different dialog items for the particular platforms you will find in category Target Platform.

The following standard platforms are generally available:

- Intel 386 compatible
- Motorola 68k
- Infineon C16x
- PowerPC
- Intel StrongARM
- MIPS
- Hitachi SH
- 8051

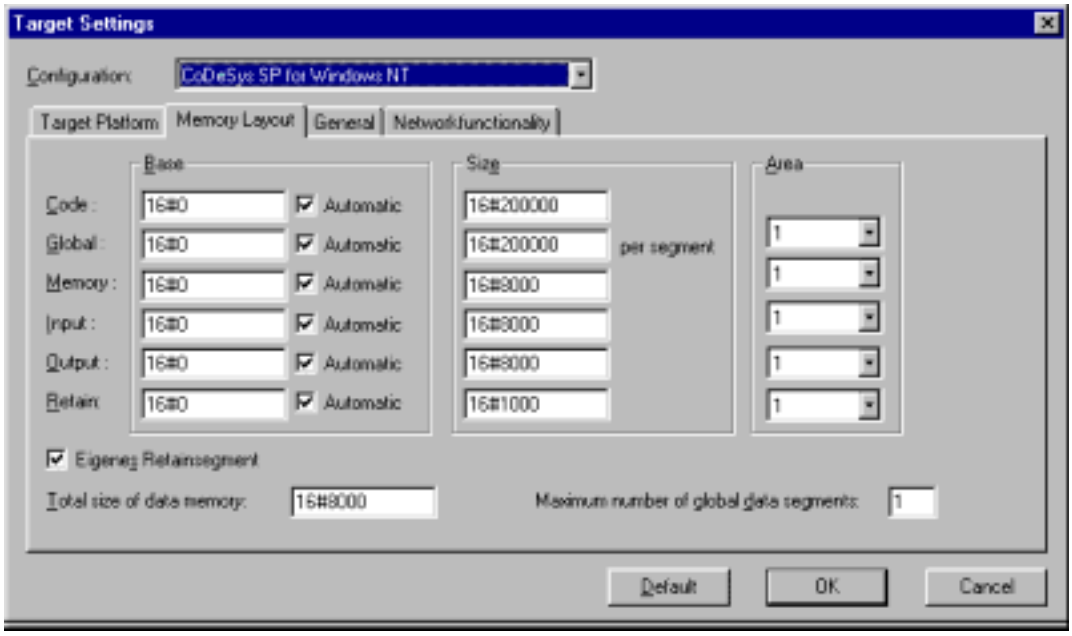
Target system 'Intel 386 compatible'

Target system 'Intel 386 compatible', Target Platform



Dialog item	Meaning
Platform	Type of the target-system
Support float processor	if activated: FPU-commands are generated for floating point operations
Debugging in multitasking environment	if activated: additional code is generated, which permits debugging in multitasking environments
Optimized jumps	if activated: optimized conditional jumps after compare operations; faster + less code (especially on 386/486); Lines containing conditions before jumps will be displayed in grey color in flow control mode
Optimized operations with constants	Optimized operations with constants (A = A + 1, A < 500 etc.); Faster + less code (especially on 386/486); Constants will be monitored in grey color in flow control mode
Optimized Loadoperations	No load operations will be executed at multiple access on a variable/constant; Faster + less code

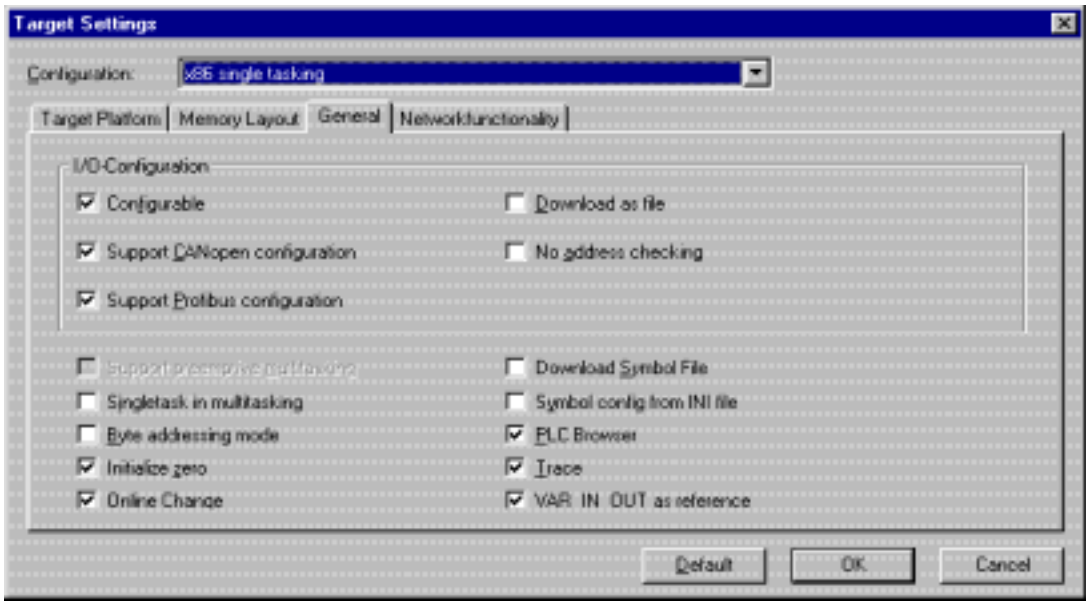
Target system 'Intel 386 compatible', Memory Layout



Dialog item	Meaning
Base (Code)	Automatic activated: Code segment is automatically allocated Automatic inactivated: Code segment lies on the given absolute address
Base (Global)	Automatic activated: Data segment (global data) are automatically allocated to the area in question Automatic inactivated: Data segment (global data) lies on the given absolute address
Base (Memory)	Automatic activated: Flags are automatically allocated to the area in question Automatic inactivated: Flag segment lies on the given absolute address
Base (Input)	Automatic activated: Input process image is automatically allocated to the area in question Automatic inactivated: Input process image lies on the given absolute address
Base (Output)	Automatic activated: Output process image is automatically allocated to the area in question Automatic inactivated: Output process image lies on the given absolute address
Base (Retain)	Automatic activated: Retentive data are automatically allocated to the area in question Automatic inactivated: Output process image lies on the given absolute address
Area (Code)	Segment number of the Data segment (Code);
Area (Global)	Segment number of the Data segment (global data);
Area (Memory)	Segment number of the Flag segment;
Area (Input)	Segment number of the Input Process Image
Area (Output)	Segment number of the Output Process Image
Area (Retain)	Segment number of the retentive data
Size (Code)	Size of the Code segment
Size pro Segment (Global)	Size of the Data segment
Size (Memory)	Size of the flag segment
Size (Input)	Size of the Input process image
Size (Output)	Size of the Output process image

Size (Retain)	Size of the Segment for retentive data
Total size of data memory	Total memory data size
Own retain segment	if activated: Retentive data are allocated to in separate segment
Total size of data memory	Total size of data memory
Maximum number of global data segments	Maximum number of global data segments
Maximum number of POUs	Maximum number of POUs in the project

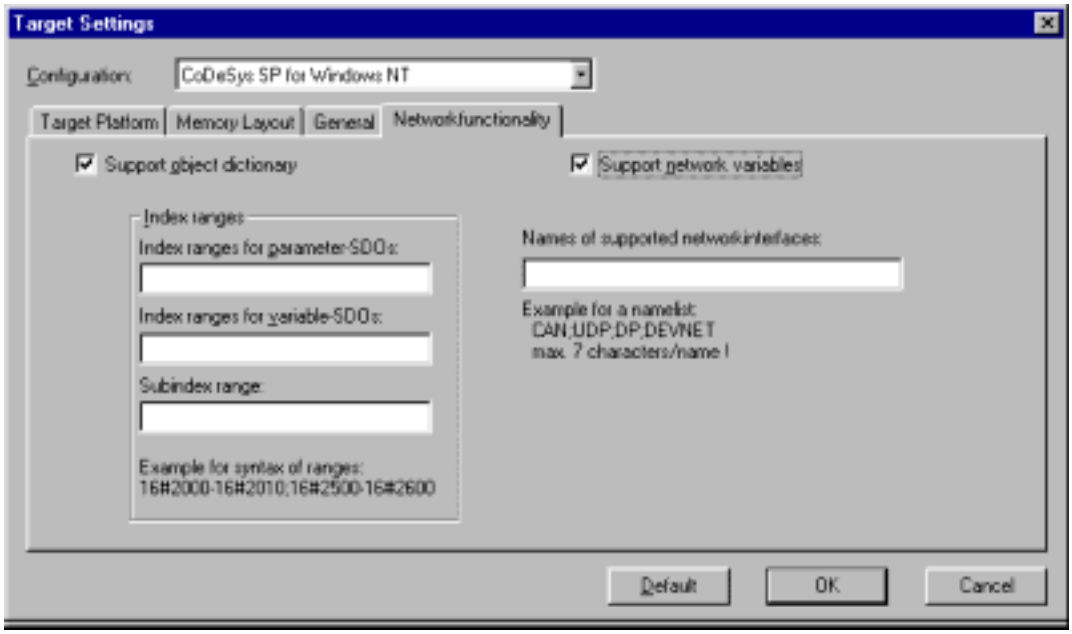
Target system 'Intel 386 compatible', General



Dialog item	Meaning
Configurable	if activated: Support configurable I/O configurations and load configuration description into the controller
Support CANopen configuration	if activated: Support CANopen configuration and load configuration description into the controller
Support Profibus configuration	if activated: Support Profibus configuration and load configuration description into the controller
Support preemptive multitasking	if activated: Support Task configuration and load task description into the controller
Download as file	if activated: I/O description is downloaded in file format
No address checking	**1=At compile the IEC addresses are not checked**
Online Change	if activated: Online Change functionality
Singletask in multitasking	not yet implemented
Byte-addressing mode	if activated: byte addressing mode (e.g. var1 AT %QD4 is written in address %QB4)
Initialize zero	if activated: General initialisation with zero
Download Symbol File	if activated: If a symbol file has been created it will be downloaded
Symbolkonfiguration aus	if activated: The parameters for the symbol configuration are not read from

INI-Datei	the project options dialog, but from the codesys.ini file, resp. from another file which is referenced in the codesys.ini
PLC-Browser	if activated: PLC Browser functionality activated
Trace	if activated: Trace functionality activated
Web visualization	if activated: All visualization objects of the project are compiled for the usage as Web visualization objects
Target visualization	if activated: All visualization objects of the project are compiled for the usage as Target visualization objects
VAR_IN_OUT by reference	if activated: At a function call the VAR_IN_OUT variables are called by reference (pointer); therefore no constants can be assigned and no read/write access is possible from outside the function block.
Initialize Inputs	if not activated: For optimizing reasons no init code will be generated for the inputs declared with "AT %IX" (-> undefined values until the 1. bus cycle !)
Autmatic bootprojekt load	if activated: After a download automatically a bootproject will be created from the new program and will be sent to the controller.
Softmotion	if activated: The SoftMotion functionality will be activated and thus is available in the Resources tab (CNC-Programmliste, Kurvenscheiben).

Target system 'Intel 386 compatible', Networkfunctionality



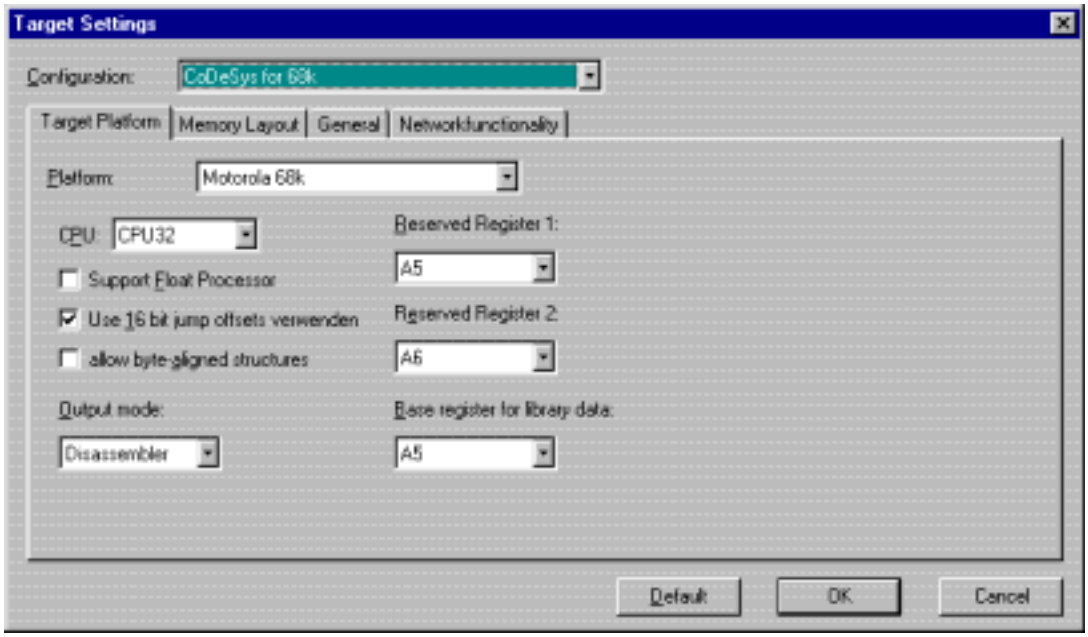
Dialog item	Meaning
Support Parameter-Manager	If this option is selected the entry ‘Parameter-Manager ’ appears in the Resources tab. Here it is possible to create an Object Dictionary for variables and parameters, which enable the targeted, active data exchange with other controllers
Support network variables	If this option is selected network variables can be used, which enable the automatic data exchange in the network

Names of supported networkinterfaces	List of the supported network systems, e.g.: CAN; UDP; DP
Index ranges for parameter SDOs	Index Range for Parameter Object Dictionaries (see Resources, OD Editor)
Index-ranges for variable SDOs	Index Range for Variables Object Dictionaries (see Resources, OD Editor)
Subindex range	Subindex Range within the above mentioned Index ranges for Parameter and Variable Ods (see Resources, OD Editor)

Target system Motorola 68K

For tabs Memory Layout, General, Networkfunctionality see 'Intel 386 compatible'.

Target Platform:

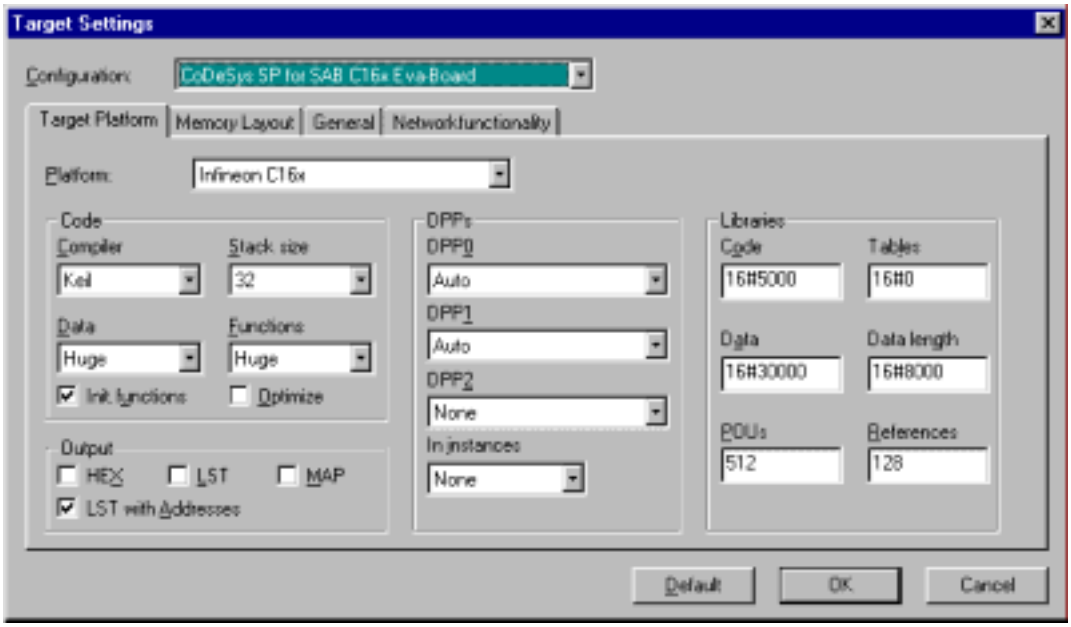


Dialog item	Meaning
Platform	Target type
CPU	Variant of the 68k CPU: basic version 68000 or CPU32 and larger
Support Float Processor	if activated: FPU-commands are generated for floating point operations
Use 16 bit jump offsets	ir activated: Jumps for evaluating Boolean expressions work with relative 16 bit Offsets (more complex expressions are possible, but larger code) if not activated: 8 bit Offsets are used
allow byte-aligned structures	if activated: only addressing even addresses if not activated: addressing of odd addresses also possible
Reserved Register 1	A2,A4,A5,A6: The indicated address register is reserved and not used If None: it can be used by the code generator
Reserved Register 2	Additional reserved address register. The indicated address register is reserved and not used If „None“ it can be used by the code generator
Base register for library data	Register for addressing static data within C libraries (before calling up library functions it is loaded with the address of free memory). If „None“ A5 is used as pre-set value.
Output-Mode	Nothing = no output Assembler = During compiling a file „code68k.hex“ is created in the compiling directory (Setting under „Project/Options/Directories“). It contains the generated Assembler Code. Disassembler = In addition to 1 the file contains the Disassembler Code

Target system Infineon C16x

For tabs Memory Layout, General, Networkfunctionality see 'Intel 386 compatible'.

Target Platform:



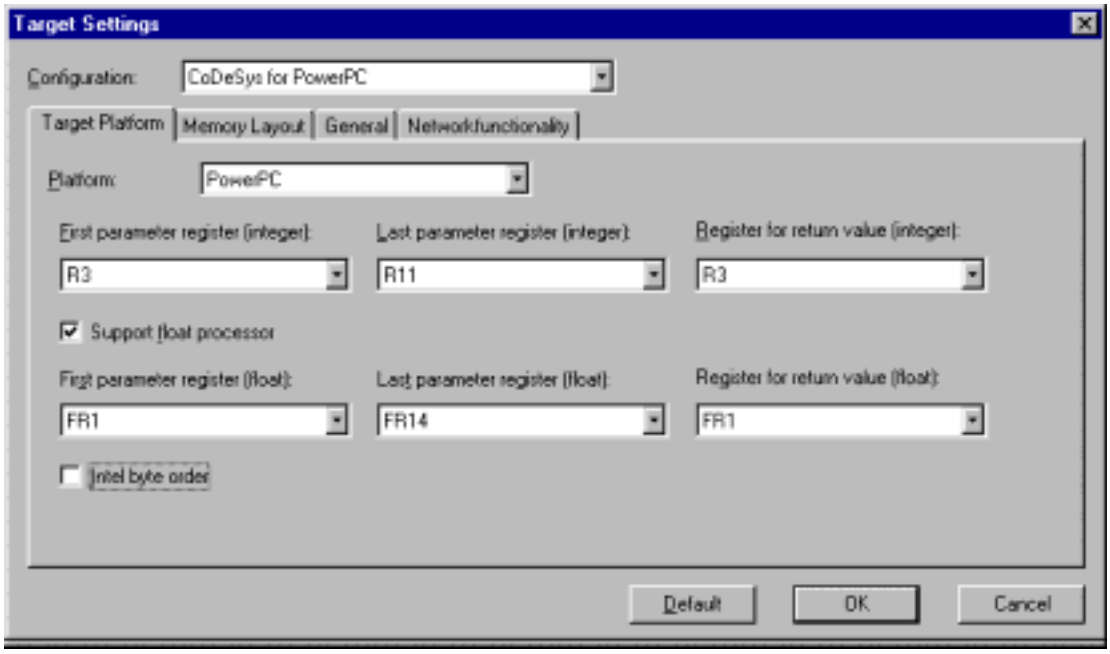
Dialog item	Meaning
Platform	Target type
Code / Compiler:	Compiler used during compiling of the target-system and the libraries (on account of C calling conventions)
Code / Stack size	Maximum call depth (nesting)
Code / Data	Memory model for data
Code / Functions	Memory model for code
Initialize functions	if activated: Functions contain initialisation code for local variables
Output HEX-File	if activated: Output of a Hex-Dump of the code
Output BIN-File	if activated: Output of a binary file of the code
Output MAP	if activated: a MAP-file of the code will be created
Output LST	if activated: a list-file of the code will be created
Output LST, with addresses	if activated: a list of the code addresses will be created
DPPs / DPP0..DPP2 In Instances	Data Page Pointers are set DPP for DPP0, DPP1, DPP2 DPP for short addressing of function block Instances
Libraries / Code Tables Data Data length POUs References	Settings for libraries: Start addresses for code, tables, data, data length, blocks, references

Target systems Intel StrongARM und Power PC

The dialog items for these two target systems are identic.

For tabs Memory Layout, General, Networkfunctionality see 'Intel 386 compatible'.

Target Platform:

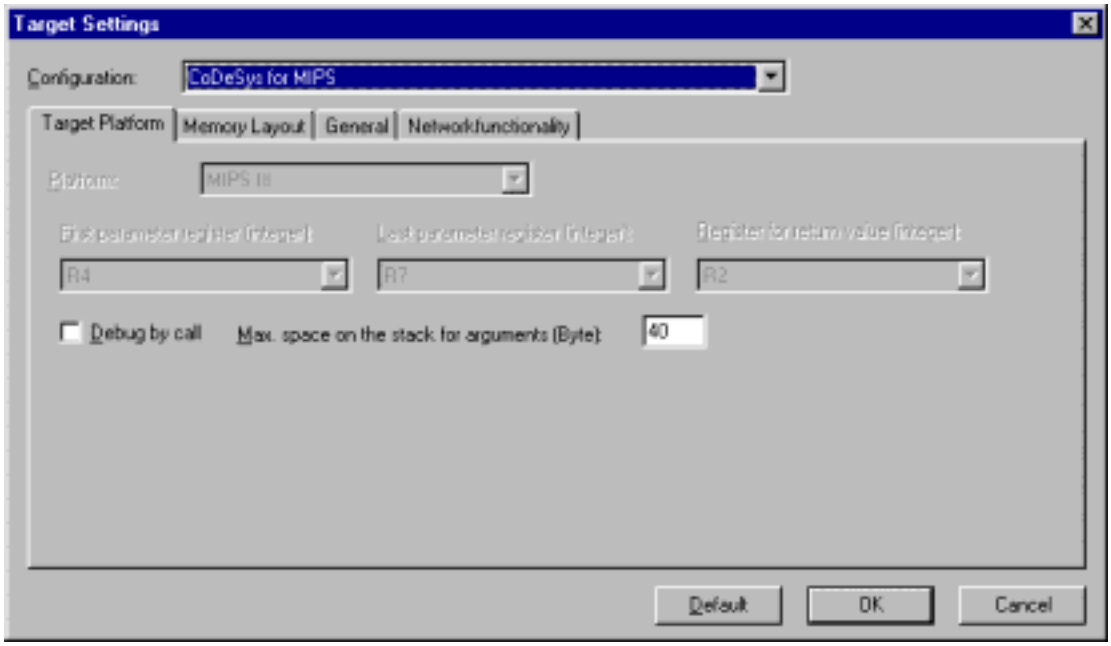


Dialog item	Meaning
Platform	Target type
Support float processor	if activated: FPU commands are generated for floating point operations
First parameter Register (integer)	Register for first Integer Parameter of C function calls (range OS dependent)
Last parameter Register (Integer)	Register for last Integer Parametersof C function calls (range OS dependent)
Register for return values (Integer)	Register for Integer return value of C function calls (range OS dependent)
First parameter Register (Float):	Register for the first Float Parameter of C function calls (range OS dependent)
Last parameter Register (Float):	Register for the last Float Parameter of C function calls (range OS dependent)
Register for return value (Float):	Register for returned Float Value
Intel byte order	if activated: Addressing as per Intel address scheme

Target system MIPS

For tabs Memory Layout, General, Networkfunctionality see 'Intel 386 compatible'.

Target Platform:

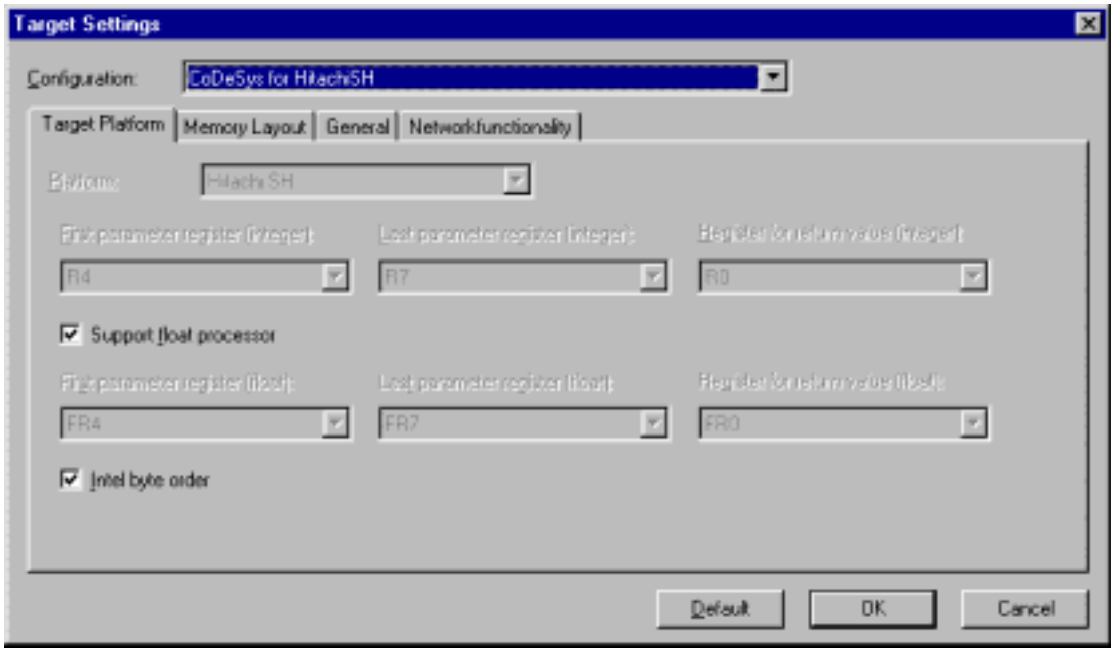


Dialog item	Meaning
Platform	Target type
First parameter Register (integer)	Register for first Integer Parameter, range is dependent on OS
Last parameter Register (Integer)	Register for last Integer Parameters, range is dependent on OS
Register for return values (Integer)	Register for Integer return values, , range is dependent on OS
Max. space on the stack for arguments (Byte):	Dependent on OS: Maximum size (Bytes) of arguments which can be handed over in the project on the stack

Target system Hitachi SH

For tabs Memory Layout, General, Networkfunctionality see 'Intel 386 compatible'.

Target Platform:

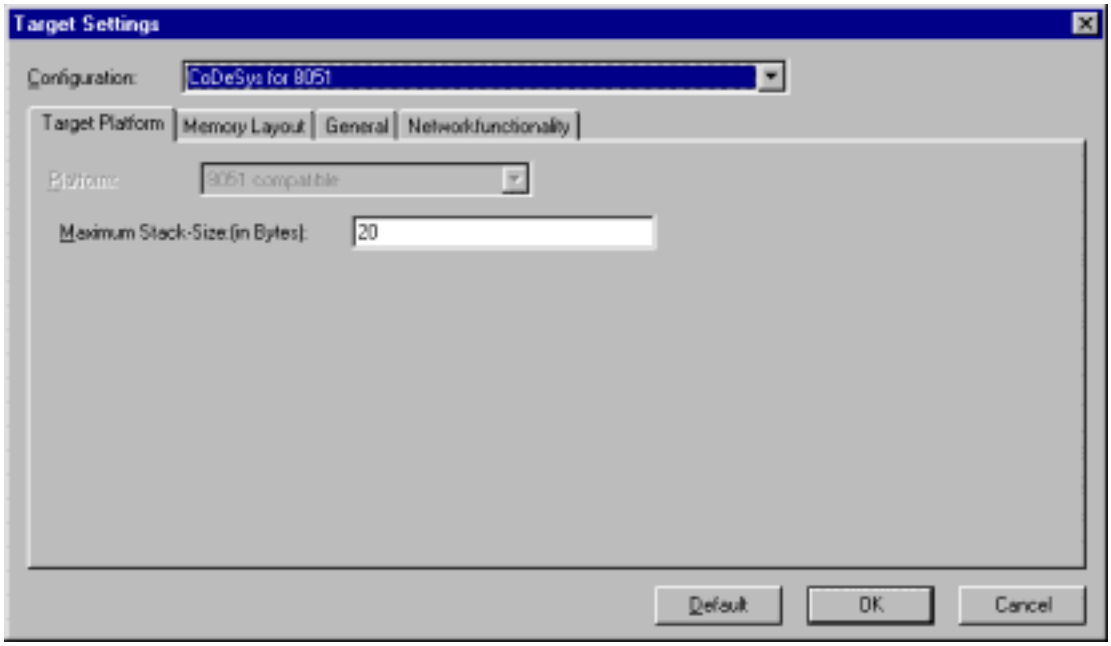


Dialog item	Meaning
Platform	Target type
First parameter Register (integer)	Register for first Integer Parameter, range is dependent on OS
Last parameter Register (Integer)	Register for last Integer Parameters, range is dependent on OS
Register for return values (Integer)	Register for Integer return values, , range is dependent on OS
Support float processor	if activated: FPU commands will be generated for floating point operations

Target system 8051 compatible

For tabs Memory Layout, General, Networkfunctionality see 'Intel 386 compatible'.

Target Platform:



Dialog item	Meaning
Platform	Target type
Maximum Stack-Size:(in Bytes)	Maximum stack size (number of Bytes)

Appendix J: Overview CoDeSys Operators and Library Elements

The table shown below shows the **operators** in ST and IL with the available modifiers in IL.

Take note that for the 'IL operator' column: Only the line in which the operator is used will be displayed. A prerequisite is that the (first) required operand have been successfully loaded in the preceding line (e.g. LD in).

The '**Mod. IL**' column shows the possible modifiers in IL:

C The command is only executed if the result of the preceding expression is TRUE.

N **for JMPC, CALC, RETC**: The command is only executed if the result of the preceding expression is FALSE.

N otherwise: negation of the operand (not of the accumulator)

(Operator enclosed in brackets: only after the closing bracket is reached will the operation preceding the brackets be carried out.

Please obtain a detailed description of usage from the appropriate Appendices.

Operators in CoDeSys:

in ST	in AWL	Mod. AWL	Description
'			String delimiters (e.g. 'string1')
.. **[ ]**			Size of Array range (e.g. ARRAY[0..3] OF INT)
:			Delimiter between Operand and Typ in a declaration (e.g. var1 : INT;)
;			Termination of instruction (e.g. a:=var1;)
^			Dereferenced Pointer (e.g. pointer1^)
	LD var1	N	Load value of var1 in buffer
:=	**ST var1**	N	Store actual result to var1
	S boolvar		Set boolean operand boolvar exactly then to TRUE, when the actual result is TRUE
	R boolvar		Set boolean operand boolvar exactly then to FALSE, when the actual result is TRUE
	JMP label	**CN**	Jump to label
<Program name>	**CAL** prog1	**CN**	Call program prog1
<Instance name>	**CAL** inst1	**CN**	Call function block instance inst1
<Fctname>(vx, vy,..)	**<Fctname> vx, vy**	**CN**	Call function fctname and transmit variables vx, vy

in ST	in AWL	Mod. AWL	Description
RETURN	**RET**	CN	Leave POU and go back to caller
	(		The value following the bracket is handled as operand, the operation before the bracket is not executed before the expression in the brackets.
	)		Now execute the operation which has been set back
AND	**AND**	N,(	Bitwise AND
OR	**OR**	N,(	Bitwise OR
XOR	**XOR**	N,(	Bitwise exclusive OR
NOT	**NOT**		Bitweise NOT
+	**ADD**	(	Addition
-	**SUB**	(	Subtraction
*	**MUL**	(	Multiplication
/	**DIV**	(	Division
>	**GT**	(	Greater than
>=	**GE**	(	Greater or equal
=	**EQ**	(	Equal
<>	**NE**	(	Not equal
<=	**LE**	(	Less or equal
<	**LT**	(	Less than
MOD(in)	**MOD**		Modulo Division
INDEXOF(in)	**INDEXOF**		Internal index of POU in1; [INT]
SIZEOF(in)	**SIZEOF**		Number of bytes required for the given data type of in
SHL(K,in)	**SHL**		Bitwise left-shift of operator in by K
SHR(K,in)	**SHR**		Bitwise right-shift of operator in by K
ROL(K,in)	**ROL**		Bitwise rotation to the left of operator in by K
ROR(K,in)	**ROR**		Bitwise rotation to the right of operator in by K
SEL(G,in0,in1)	**SEL**		Binary selection between 2 operands in0 (G is FALSE) and in1 (G is TRUE)
MAX(in0,in1)	**MAX**		Returns the greater of 2 values
MIN(in0,in1)	**MIN**		Returns the lesser of 2 values in0 and in1
LIMIT(MIN,in,Max)	**LIMIT**		Limits the value range (in is set back to MIN or MAX in case of exceeding the range)
	JMP label	**CN**	Jump to label
<Program name>	**CAL** prog1	**CN**	Call program prog1
<Instance name>	**CAL** inst1	**CN**	Call function block instance inst1
<Fctname>(vx, vy,..)	**<Fctname> vx, vy**	**CN**	Call function fctname and transmit variables vx, vy
RETURN	**RET**	CN	Leave POU and go back to caller
	(		The value following the bracket is handled as operand, the operation before the bracket is not executed before the expression in the brackets.
	)		Now execute the operation which has been set back
AND	**AND**	N,(	Bitwise AND
OR	**OR**	N,(	Bitwise OR
XOR	**XOR**	N,(	Bitwise exclusive OR

in ST	in AWL	Mod. AWL	Description
NOT	**NOT**		Bitweise NOT
+	**ADD**	(	Addition
-	**SUB**	(	Subtraction
*	**MUL**	(	Multiplication
/	**DIV**	(	Division
>	**GT**	(	Greater than
>=	**GE**	(	Greater or equal
=	**EQ**	(	Equal
<>	**NE**	(	Not equal
<=	**LE**	(	Less or equal
<	**LT**	(	Less than
MOD(in)	**MOD**		Modulo Division
INDEXOF(in)	**INDEXOF**		Internal index of POU in1; [INT]
SIZEOF(in)	**SIZEOF**		Number of bytes required for the given data type of in
SHL(K,in)	**SHL**		Bitwise left-shift of operator in by K
SHR(K,in)	**SHR**		Bitwise right-shift of operator in by K
ROL(K,in)	**ROL**		Bitwise rotation to the left of operator in by K
ROR(K,in)	**ROR**		Bitwise rotation to the right of operator in by K
SEL(G,in0,in1)	**SEL**		Binary selection between 2 operands in0 (G is FALSE) and in1 (G is TRUE)
MAX(in0,in1)	**MAX**		Returns the greater of 2 values
MIN(in0,in1)	**MIN**		Returns the lesser of 2 values in0 and in1
LIMIT(MIN,in,Max)	**LIMIT**		Limits the value range (in is set back to MIN or MAX in case of exceeding the range)
MUX(K,in0,...in_n)	**MUX**		Selectsi the Kth value out of a group of values (in0 to In_n)
ADR(in)	**ADR**		Address of the operand in [DWORD]
BOOL_TO_<type>(in)	**BOOL_TO_<type>**		Type conversion of the boolean operand
<type>_TO_BOOL(in)	**<type>_TO_BOOL**		Type conversion to BOOL
INT_TO_<type>(in)	**INT_TO_<type>**		Type conversion of an INT Operand to anotner elementary type
REAL_TO_<type>(in)	**REAL_TO_<type>**		Type conversion of an REAL operand to anotner elementary type
LREAL_TO_<type>(in)	**LREAL_TO_<type>**		Type conversion of a LREAL operand to anotner elementary type
TIME_TO_<type>(in)	**TIME_TO_<type>**		Type conversion of a TIME operand to anotner elementary type
TOD_TO_<type>(in)	**TOD_TO__<type>**		Type conversion of a TOD operand to anotner elementary type
DATE_TO_<type>(in)	**DATE_TO_<type>**		Type conversion of a DATE operand to anotner elementary type
DT_TO_<type>(in)	**DT_TO_<type>**		Type conversion of a DT operand to anotner elementary type

STRING_TO_<type>(in)	**STRING_TO_<type>**		Type conversion of a string operand des Operanden to anotner elementary type, in must contain valid value of desired type
TRUNC(in)	**TRUNC**		Conversion from REAL to INT
ABS(in)	**ABS**		Absolut value of operand in
SQRT(in)	**SQRT**		Square root of operand in
LN(in)	**LN**		Natural logarithm of operand in
LOG(in)	**LOG**		Logarithm of operand in, base 10
EXP(in)	**EXP**		Exponential function of operand in
SIN(in)	**SIN**		Sine of operand in
COS(in)	**COS**		Cosine of operand in
TAN(in)	**TAN**		Tangent of operand in
ASIN(in)	**ASIN**		Arc sine of operand in
ACOS(in)	**ACOS**		Arc cosine of operand in
ATAN(in)	**ATAN**		Arc tangent of operand in
EXPT(in,expt)	**EXPT** expt		Exponentation of operand in with expt

Elements of the Standard.lib:

Operator ST	Operator AWL	Description
LEN(in)	**LEN**	String length of operand in
LEFT(str,size)	**LEFT**	Left inital string of given size of string str
RIGHT(str,size)	**RIGHT**	Right initial string of given size of string str
MID(str,size)	**MID**	Partial string of str of given size
CONCAT('str1','str2')	**CONCAT** 'str2'	Concatenation of two subsequent strings
INSERT('str1','str2',pos)	**INSERT** 'str2',p	Insert string str1 in String str2 at position pos
DELETE('str1',len,pos)	**DELETE** len,pos	Delete partial string (length len), start at position pos of str1
REPLACE('str1','str2',len,pos)	**REPLACE** 'str2',len,pos	Replace partial string of lenght len by str2, start at position pos of str1
FIND('str1','str2')	**FIND** 'str2'	Search for partial string str2 in str1
SR	**SR**	Bistable FB is set dominant[1]
RS	**RS**	Bistable FB is set back[1]
SEMA	**SEMA**	FB: Software Semaphor (interruptable) [1]
R_TRIG	**R_TRIG**	FB: rising edge is detected[1]
F_TRIG	**F_TRIG**	FB: falling edge is detected[1]
CTU	**CTU**	FB: Counts upv[1]
CTD	**CTD**	FB: Counts down[1]
CTUD	**CTUD**	FB: Counts up and down[1]
TP	**TP**	FB: trigger[1]
TON	**TON**	FB: Einschaltverzögerung[1]
TOF	**TOF**	FB: Ausschaltverzögerung[1]
RTC	**RTC**	FB: Laufzeit-Uhr[1]
BCD_TO_INT	**BCD_TO_INT**	Conversion of a Byte: BCD to INT format[2]
INT_TO_BCD	**INT_TO_BCD**	Converstion of a Byte: INT to BCD format[2]
EXTRACT(in,n)	**EXTRACT n**	The n-th bit of DWORD in is returned in BOOl[2]

PACK	**PACK**	Up to 8 bits are packed into a byte[2]
PUTBIT	**PUTBIT**	A bit of a DWORD is set to a certain value[2]
UNPACK	**UNPACK**	A Byte is returned as single bits[2]

Elements of the Util.lib:

	Description
DERIVATIVE	Local derivation[2]
INTEGRAL	Integral[2]
STATISTICS_INT	Min.,Max, Average values in INT format[2]
STATISTICS_REAL	Min.,Max, Average in REAL format[2]
VARIANCE	Variance[2]
PD	PD controller[2]
PID	PID controller[2]
BLINK	Pulsating signal[2]
GEN	Periodic functions[2]
CHARCURVE	linear functions[2]
RAMP_INT	Limiting ascendance of descendance of the function beeing fed (INT)[2]
RAMP_REAL	Limiting ascendance of descendance of the function beeing fed (REAL)[2]
HYSTERESIS	Hysteresis[2]
LIMITALARM	Watches whether input value exceeds limits of a defined range[2]
FileOpen	Open a file[3]
FileRead	Read a file [3]
FileWrite	Write a file [3]
FileDelete	Delete a file [3]
FileClose	Close a file [3]

Appendix K: Compiler Errors and Warnings

SEQWarnings

1100

"Unknown function '<name>' in library."
An external library is used. Please check, whether all functions, which are defined in the .hex file, are also defined in the .lib file.

1101

"Unresolved symbol '<Symbol>'."
The code generator expects a POU with the name <Symbol>. It is not defined in the project. Define a function/program with this name.

1102

"Invalid interface for symbol '<Symbol>'."
The code generator expects a function with the name <Symbol> and exactly one scalar input, or a program with the name <Symbol> and no input or output.

1103

"The constant '<name>' at code address '<address>' overwrites a 16K page boundary!"
A string constant exceeds the 16K page boundary. The system cannot handle this. It depends on the runtime system whether the problem could be avoided by an entry in the target file. Please contact the PLC manufacturer.

1200

"Task '<name>', call of Access variables in the parameter list are not updated"
Variables, which are only used at a function block call in the task configuration, will not be listed in the cross reference list.

1300

"File not found '<name>'"
The file, to which the global variable object is pointing, does not exist. Please check the path.

1301

"Analyze-Library not found! Code for analyzation will not be generated."
The analyze function is used, but the library analyzation.lib is missing. Add the library in the library manager.

1302

"New externally referenced functions inserted. Online Change is therefore no longer possible!"
Since the last download you have linked a library containing functions which are not yet referenced in the runtime system. For this reason you have to download the complete project.

1400

"Unknown Pragma '<name>' is ignored!"
This pragma is not supported by the compiler. See keyword ‘pragma’ for supported directives.

1401

"The struct '<name>' does not contain any elements."
The structure does not contain any elements, but variables of this type allocate 1 Byte of memory.

1410

"'RETAIN' and 'PERSISTENT' do not have any effect in functions"
Remanent variables are handled like normal local variables.

1411

"Variable '<name>' in the variable configuration isn't updated in any task"

```
The top level instance of the variable is not referenced by a call in any task.
Thus it will not be copied from the process image.
Example:
      Variable Configuration:
       VAR_CONFIG
            plc_prg.aprg.ainst.in AT %IB0 : INT;
       END_VAR
      plc_prg:
       index := INDEXOF(aprg);
The program aprg is referenced but not called. Thus plc_prg.aprg.ainst.in never
will get the actual value of %IB0.
```

1500

"Expression contains no assignment. No code was generated."

The result of this expression is not used. For this reason there is no code generated for the whole expression.

1501

"String constant passed as 'VAR_IN_OUT': '<name>' must not be overwritten!"

The constant may not be written within the POU, because there no size check is possible.

1502

"Variable '<name>' has the same name as a POU. The POU will not be called!"

Rename variable or POU.
Beispiel:

```
PROGRAM a
...
VAR_GLOBAL
        a: INT;
END_VAR
...

a; (* Not POU a is called but variable a is loaded. *)
```

1503

"The POU '<name>' has no outputs. Box result is set to 'TRUE'."

The Output pin of a POU which has no outputs, is connected in FBD or KOP. The assignment automatically gets the value TRUE.

1504

"'<name>' ('<number>'): Statement may not be executed due to the evaluation of the logical expression"

Eventually not all branches of the logic expression will be executed.
Example:
IF a AND funct(TRUE) THEN ....
If a has is FALSE then funct will not be called.

1505

"Side effect in '<name>'! Branch is probably not executed !"

The first input of the POU is FALSE, for this reason the side branch, which may come in at the second input, will not be executed.

1506

"Variable '<name>' has the same name as a local action. The action will not be called!"

Rename the variable or the action.

1600

"Open DB unclear (generated code may be erroneous)."

The original Siemens program does not tell, which POU is openend.

1700

"Input box without assignment."
An input box is used in CFC which has no assignment. For this no code will be generated.

1800

"<name>(element #<element number>): Invalid watchexpression '%s'"
The visualization element contains an expression which cannot be monitored. Check variable name and placeholder replacements.

1801

"'<name> (number): No Input on Expression '<name>' possible"
In the configuration of the visualization object at field input a composed expression is used. Replace this by a single variable.

1802

"<Visualization object>(Element number): Bitmap '%s' was not found"
Make sure, that an external bitmap-file is available in that path which is defined in the visualization configuration dialog.

1900

"POU '<name>' (main routine) is not available in the library"
The Start-POU (z.B. PLC_PRG) will not be available, when the project is used as library.

1901

"Access Variables and Variable Configurations are not saved in a library!"
Access variables and variable configuration are not stored in the library.

1902

"'<name>': is no Library for the current machine type!"
The .obj file of the lib was generated for another device.

1903

"<name>: is no valid Library"
The file does not have the format requested for the actual target.

1904

"The constant '<Name>' hides a constant of the same name in a library"
In your project you have defined a constant which has the same name like one which is defined in a linked library. The library variable will be overwritten !

Compiler Errors

3100

"Code too large. Maximum size: '<number>' Byte (<number>K)"
The maximum program size is exceeded. Reduce project size.

3101

"Total data too large. Maximum size: '<number>' Byte (<number>K)"
Memory is exceeded. Reduce data usage of the application.

3110

"Fehler in Bibliotheks-Datei '<name>'."
The .hex file is not in INTEL Hex format.

3111

"Library '<name>' is too large. Maximum size: 64K"
The .hex file exceeds the set maximum size.

3112

"Nonrelocatable instruction in library."
The .hex file contains a nonrelocatable instruction. The library code cannot be linked.

3113

"Library code overwrites function tables."
The ranges for code and function tables are overlapping.

3114

"Library uses more than one segment."
The tables and the code in the .hex file use more than one segment.

3115

"Unable to assign constant to VAR_IN_OUT. Incompatible data types."
The internal pointer format for string constants cannot get converted to the internal pointer format of VAR_IN_OUT, because the data are set "near" but the string constants are set " huge" or "far". If possible change these target settings.

3116

"Function tables overwrite library code or a segment boundary."
Code 166x: The external library cannot be used with the current target settings. These must be adapted resp. the library must be rebuilt with appropriate settings.

3120

"Current code-segment exceeds 64K."
The currently generated code is bigger than 64K. Eventually to much initializing code is created.

3121

"POU too large."
A POU may not exceed the size of 64K.

3122

"Initialisation too large. Maximum size: 64K"
The initialization code for a function or a structure POU may not exceed 64K.

3123

"Data segment too large: segment '<Number>%s', size <size> bytes (maximum <number> bytes)"
Please contact your manufacturer.

3130

"User-Stack too small: '<number>' DWORD needed, '<number>' DWORD available."
The nesting depth of the POU calls is to big. Enter a higher stack size in the target settings or compile build project without option ‚Debug' (can be set in dialog 'Project' 'Options' 'Build').

3131

"User-Stack too small: '<number>' WORD needed, '<number>' WORD available."
Please contact the PLC manufacturer.

3132

"System-Stack too small: '<number>' WORD needed, '<number>' WORD available."
Please contact the PLC manufacturer.

3150

"Parameter <number> of function '<name>': Cannot pass the result of a IEC-function as string parameter to a C-function."

Use a intermediate variable, to which the result of the IEC function is assigned.

3160

"Can't open library file '<name>'."
A library <name> is included in the library manager for this project, but the library file does not exist at the given path.

3161

"Library '<name>' contains no codesegment"
A .obj file of a library at least must contain one C function. Insert a dummy function in the .obj file, which is not defined in the .lib file.

3162

"Could not resolve reference in Library '<name>'(Symbol '<name>', Class '<name>', Type '<name>')"
The .obj file contains a not resolvable reference to another symbol. Please check-the settings of the C-Compiler.

3163

"Unknown reference type in Library '<name>' (Symbol '<name>' , Class '<name>' , Type '<name>')"
The .obj file contains a reference type, which is not resolvable by the code generator. Please check-the settings of the C-Compiler.

3200

"<name> (<name>): Boolean expression to complex"
The temporary memory of the target system is insufficient for the size of the expression. Divide up the expression into several partial expressions thereby using assignments to intermediate variables.

3201

"<name> (<network>): A network must not result in more than 512 bytes of code"
Internal jumps can not be resolved. Activate option "Use 16 bit Sprungoffsets" in the 68k target settings.

3202

"Stack overrun with nested string/array/structure function calls"
A nested function call CONCAT(x, f(i)) is used. This can lead to data loss. Divde up the call into two expressions.

3203

"Expression too complex (too many used adress registers)."
Divide up the assignment in several expressions.

3204

"A jump exceeds 32k Bytes"
Jump distances may not be bigger than 32767 bytes.

3205

"Internal Error: Too many constant strings"
In a POU there at the most 3000 string constants may be used.

3206

"Function block data exceeds maximal size"
A function block may produce maximum 32767 Bytes of code.

3207

"Array optimization"
The optimization of the array accesses failed because during index calculation a function has been called.

3208

"Conversion not implemented yet"
A conversion function is used, which is not implemented for the actual code generator.

3209

"Operator not implemented"
A operator is used, which is not implemented for this data type and the actual code generator. MIN(string1,string2).

3210

"Function '<name>' not found"
A function is called, which is not available in the project.

3211

"Max string usage exceeded"
A variable of type string can be used in one expression 10 times at the most.

3212

"Wrong library order at POU <POU name>"
The order of libraries for this POU does not match with that in the cslib.hex file. Correct the order accordingly. (only for 68K targets, if the checking option is activated in the target file.)

3250

"Real not supported for 8 Bit Controller"
The target is currently not supported.

3251

"date of day types are not supported for 8 Bit Controller"
The target is currently not supported.

3252

"size of stack exeeds <number> bytes"
The target is currently not supported.

3253

"Could not find hex file: '<name>' "
The target is currently not supported.

3254

"Call to external library function could not be resolved."
The target is currently not supported.

3255

"Pointers are not supported for 8 bit controllers."
Avoid using pointers in your program to get it running on the 8 bit system.

3400

"An error occured during import of Access variables"
The .exp file contains an incorrect access variables section.

3401

"An error occured during import of variable configuration"
The .exp file contains an incorrect configuration variables section.

3402

"An error occured during import of global variables"
The .exp file contains an incorrect global variables section.

3403

"Could not import <name>"
The section for object <name> in the .exp file is not correct.

3404

"An error occured during import of task configuration"

The section for the task configuration the .exp file is not correct.

3405

"An error occured during import of PLC configuration"
The section for the PLC configuration in the .exp file is not correct.

3406

"Two steps with the name '<name'. Second step not imported."
The section for the SFC POU in the .exp file contains two steps with equal names. Rename one of the steps in the export file.

3407

"Predecessor step '<name>' not found"
The step <name> is missing in the .exp file.

3408

"Successor step '<name>' not found"
The step <name> is missing in the .exp file.

3409

"No successing transition for step '<´name>' "
In the .exp file a transition is missing, which requires step <name> as preceeding step.

3410

"No successing step for transition '<name>'"
In the .exp file a step is missing which requires the transition <name> as preceeding condition.

3411

"Step '<name>' not reachable from initial step"
In the .exp file the connection between step <name> and the initial step is missing.

3412

"Macro '<name>' not imported"
Check the export file.

3450

"PDO'<PDO-name>': Missing COB-Id!"
Click on the button ‚Properties' in the PLC configuration dialog for the module and enter a COB ID for the PDO <PDO Name>.

3451

"Error during load: EDS-File '<name>' could not be found, but is referenced in hardware configuration!"
Eventually the device file needed for the CAN configuration is not in the correct directory. Check the directory setting for configuration files in ‚Project' 'Options' 'Directories'.

3452

"The module '<name>' couldn't be created!"
The device file for module <name> does not fit to the current configuration. Eventually it has been modified since the configuration has been set up in CoDeSys or it is corrupted.

3453

"The channel '<name>' couldn't be created!"
The device file for channel <name> does not fit to the current configuration. Eventually it has been modified since the configuration has been set up in CoDeSys or it is corrupted.

3454

"The address '<name>' points to an used memory!"

Option 'Check for overlapping addresses' is activated in the dialog ‚Settings' of the PLC configuration and an overlap has been detected. Regard, that the area check is based on the size which results of the data types of the modules, not on the size which is given by the entry ‚size' in the configuration file.

3455

"Error during load: GSD-File '<name>' could not be found, but is referenced in hardware configuration!"
Eventually the device file required by the Profibus configuration is not in the correct directory. . Check the directory setting for configuration files in ‚Project' 'Options' 'Directories'.

3456

"The profibus device '<name>' couldn't be created!"
The device file for module <name> does not fit to the current configuration. Eventually it has been modified since the configuration has been set up in CoDeSys or it is corrupted.

3457

"Error in module description!"
Please check the device file of this module.

3458

"The PLC-Configuration couldn't be created! Check the configuration files."
Check if all required configuration and device files are available in the correct path (see defined compile directory in 'Project' 'Options' /Directories)

3500

"No 'VAR_CONFIG' for '<name>'"
Insert a declaration for this variable in the global variable list which contains the 'Variable_Configuration'.

3501

"No address in 'VAR_CONFIG' for '<name>'."
Assign an address to this variable in the global variable list which contains the 'Variable_Configuration'.

3502

"Wrong data type for '<name>' in 'VAR_CONFIG"
In the global variables list which contains the ‚Variable_Configuration' the variable is declared with a different data type than in the POU.

3503

"Wrong data type for '<name>' in 'VAR_CONFIG'"
In the global variables list which contains the ‚Variable_Configuration' the variable is declared with a different address than in the POU.

3504

"Initial values are not supported for 'VAR_CONFIG"
A variable of the ‚Variable_Configuration' is declared with address and initial value. But an initial value can only be defined for input variables without address assignment.

3505

"'<name>'is no valid instance path"
The Variable_Configuration contains a nonexisting variable.

3506

"Access path expected"
In the global variable list for Access Variables the access path for a variable is not correct. Correct: <Identifier>:'<Access path>':<Type> <Access mode>

3507

"No address specification for 'VAR_ACCESS'-variables"
The global variable list for Access Variables contains an address assignment for a variable. This is not allowed.

Valid variable definition: <Identifier>:'<Access path>':<Type> <Access mode>

3550

"Duplicate definition of identifier '<name>'"
There are two tasks are defined with an identic same name. Rename one of them.

3551

"The task '<name>' must contain at least one program call"
Insert a program call or delete task.

3552

"Event variable '<name>' in task '%s' not defined"
There is an event variable set in the ‘Single’ field of the task properties dialog which is not declared globally in the project. Use another variable or define the variable globally.

3553

"Event variable '<name>' in task '%s' must be of type 'BOOL'"
Use a variable of type BOOL as event variable in the ‘Single’ field of the task properties dialog.

3554

"Task entry '<name>' must be a program or global function block instance"
In the field ‚Program call’ a function or a not defined POU is entered. Enter a valid program name.

3555

"The task entry '<name>' contains invalid parameters"
In the field ‚Append program call’ there are parameters used which do not comply with the declaration of the program POU.

3556

"Tasks are not supported by the currently selected target"
The currently defined task configuration cannot be used for the currently set target system. Change target or modify the task configuration correspondingly.

3557

"Maximum number of Tasks ('<number>') exceeded"
The currently defined number of tasks exceeds the maximum number allowed for the currently set target system. Change target or modify the task configuration correspondingly. Attention: Do not edit the XML description file of the task configuration !

3558

"Priority of task '<name>' is out of valid range between '<lower limit>' and '<upper limit>'"
The currently defined priority for the task is not valid for the currently set target system. Change target or modify the task configuration correspondingly.

3559

"Task '<name>': Interval-Tasks are not supported by the current target"
The current task configuration contains an interval task. This is not allowed by the currently set target system. Change target or modify the task configuration correspondingly.

3560

"Task '<name>': free wheeling tasks are not supported by the current target"
The current task configuration contains an free wheeling task. This is not allowed by the currently set target system. Change target or modify the task configuration correspondingly.

3561

"Task '<name>': event tasks are not supported by the current target"
The current task configuration contains event tasks which are not supported by the currently set target system. Change target or modify the task configuration correspondingly.

3562

"Task '<name>': external event tasks are not supported by the current target"
The current task configuration contains external event tasks which are not supported by the currently set target system. Change target or modify the task configuration correspondingly.

3563

"The interval of task '<name>' is out of valid range between '<lower limit>' and '<upper limit>'"
Change the interval value in the configuration dialog for the task.

3564

"The external event '<name>' of task '<name>' is not supported by the current target"
The currently set target system does not support the external event which is defined in the task configuration for this task. Change target or modify the task configuration correspondingly. Attention: Do not edit the XML description file of the task configuration !

3565

"Maximum number of event tasks ('<number>') exceeded"
The currently set target system does not allow as many event tasks as are defined at the moment. Change target or modify the task configuration correspondingly.

3566

"Maximum number of interval tasks ('<number>') exceeded"
The currently set target system does not allow as many interval tasks as defined at the moment. Change target or modify the configuration correspondingly.

3567

"Maximum number of free wheeling tasks ('<number>') exceeded"
The currently set target system does not allow as many free wheeling tasks as defined at the moment. Change target or modify the configuration correspondingly.

3568

"Maximum number of external interval tasks ('<number>') exceeded"
The currently set target system does not allow as many external interval tasks as defined at the moment. Change target or modify the configuration correspondingly.

3569

"POU '<name>' for system event '<name>' not defined"
The POU which should be called by the named system event, as defined in the task configuration, is not available in the project. Modify the task configuration correspondingly or make sure that the POU is available in the project.

3570

"The tasks '<name>' and '<name>' share the same priority"
Modify the task configuration so that each task has a different priority.

3571

"The library 'SysLibCallback' is not included in the project! System events can not be generated."
In order to create event tasks, the SysLibCallback.lib is needed. Link this library to the project in the library manager or modify the task configuration (task attributs) in that way that there is no task triggered by an event.

3600

“Implicit variables not found!”
Use command ‚Rebuild all’. If nevertheless you get the error message again please contact the PLC manufacturer.

3601

"<name> is a reserved variable name"
The given variable is declared in the project, although it is reserved for the codegenerator. Rename the variable.

3610

" '<name>' not supported"
The given feature is not supported by the current version of the programming system.

3611

"The given compile directory '<name>' is invalid"
There is an invalid directory given in the ‚Project' ‚Options' ‚Directories' for the Compile files.

3612

"Maximum number of POUs (<number>) exceeded! Compile is aborted."
Too many POUs and data types are used in the project. Modify the maximum number of POUs in the Target Settings / Memory Layout.

3613

"Build canceled"
The compile process was cancelled by the user.

3614

"Project must contain a POU named '<name>' (main routine) or a taskconfiguration"
Create an init POU of type Program (e.g. PLC_PRG) or set up a task configuration.

3615

"<name> (main routine) must be of type program"
A init POU (e.g. PLC_PRG) is used in the project which is not of type Program.

3616

"Programs musn't be implemented in external libraries"
The project which should be saved as an external library contains a program. This will not be available, when the library will be used.

3617

"Out of memory"
Increase the virtual memory capacity of your computer.

3618

"BitAccess not supported in current code generator!"
The code generator for the currently set target system does not support bit access on variables.

3700

" POU with name ‘<name>' is already in library '<name>'"
A POU name is used in the project, which is already used for a library POU. Rename the POU.

3701

"Name used in interface is not identical with POU Name"
Use command **‘Project' ‘Rename object'** to rename the POU in the object organizer, or change the name of the POU in the declaration window. There the POU name has to be placed next to one of the keywords PROGRAM, FUNCTION oder FUNCTIONBLOCK..

3702

"Overflow of identifier list"
Maximum 100 identifiers can be entered in one variable declaration.

3703

"Duplicate definition of identifier '<name>'"
Take care that there is only one identifier with the given name in the declaration part of the POU.

3704

"data recursion: "<POU 0> -> <POU 1> -> .. -> <POU 0>"
A FB instance is called, which is calling itself.

3705

"VAR_IN_OUT' not allowed in top level POU if no task configuration is available"
Set up a task configuration or make sure that no VAR_IN_OUT variables are used in PLC_PRG.

3720

"Address expected after 'AT'"
Add a valid address after the keyword AT or modify the keyword.

3721

"Only 'VAR' and 'VAR_GLOBAL' can be located to addresses"
Put the declaration to a VAR or VAR_GLOBAL declaration area.

3722

"Only 'BOOL' variables allowed on bit addresses"
Modify the address or modify the type of the variable to which the address is assigned.

3726

"Constants can not be laid on direct addresses"
Modify the address assignment correspondingly.

3727

"No array declaration allowed on this address"
Modify the address assignment correspondingly.

3728

"Invalid address: '<address>'"
This address is not supported by the PLC configuration. Check PLC configuration resp. modify address.

3729

"Invalid type '<name>' at address: '<name>' "
The type of this variable cannot be placed on the given address. Example: For a target system working with ‘alignment 2’ the following declaration is not valid: var1 AT %IB1:WORD;

3740

"Invalid type: '<name>' "
An invalid data type is used in a variable declaration.

3741

"Expecting type specification"
A keyword or an operator is used instead of a valid type identifier.

3742

"Enumeration value expected"
In the definition of the enumeration type an identifier is missing after the opening bracket or after a comma between the brackets.

3743

"Integer number expected"
Enumerations can only be initialized with numbers of type INT.

3744

"Enum constant '<name>' already defined"
Check if you have followed the following rules for the definition of enumeration values:
- Within one enum definition all values have to be unique.
- Within all global enum definitions all values have to be unique.
- Within all local enum definitions all values have to be unique.

3745

"Subranges are only allowed on Integers!”

Subrange types can only be defined resting on integer data types.

3746

"Subrange '<name>' is not compatible with Type '<name>'"
One of the limits set for the range of the subrange type is out of the range which is valid for the base type.

3747

"unknown string length: '<name>'"
There is a not valid constant used for the definition of the string length.

3748

"More than three dimensions are not allowed for arrays"
More than the allowed three dimensions are given in the definition of an array. If applicable use an ARRAY OF ARRAY.

3749

"lower bound '<name>' not defined"
There is a not defined constant used to define the lower limit for a subrange or array type.

3750

"upper bound '<name>' not defined"
There is a not defined constant used to define the upper limit for a subrange or array type.

3751

"Invalid string length '<number of characters>'"
The here defined string length exceeds the maximum value which is defined for the currently set target system.

3760

"Error in inital value"
Use an initial value which corresponds to the type definition. To change the declaration you can use the declaration dialog for variables (Shift/F2 or 'Edit"Autodeclare').

3761

"'VAR_IN_OUT' variables must not have an inital value."
Remove the initialization at the declaration of the VAR_IN_OUT variable.

3780

"'VAR', 'VAR_INPUT', 'VAR_OUTPUT' or 'VAR_IN_OUT' expected"
The first line following the name of a POU must contain one of these keywords.

3781

"'END_VAR' or identifier expected"
Enter a valid identifier of a END_VAR at the beginning of the given line in the declaration window.

3782

"Unexpected end"
In the declaration editor: Add keyword END_VAR at the end of the declaration part.
In the texteditor of the programming part: Add an instruction which terminates the last instruction sequence (e.g. END_IF).

3783

"END_STRUCT' or identifier expected"
Ensure that the type declaration is terminated correctly.

3784

"The current target doesn't support attribute <attribute name>"
The target system does not support this type of variables (e.g. RETAIN, PERSISTENT)

3800

"The global variables need too much memory. Increase the available memory in the project options."

Increase the number of segments given in the settings in dialog ‚Project' ‚Options' ‚Build'.

3801

"Die Variable '<name>' ist zu groß. (<Größe> Byte)"
The variable uses a type which is bigger than 1 data segment. The segment size is a target specific parameter and can be modified in the target settings/memory layout. If you do not find this in the current target settings, please contact your PLC manufacturer.

3802

"Out of retain memory. Variable '<name>', <number> bytes."
The memory space available for Retain variables is exhausted. The size of the memory area can be set target-specific in the target settings /memory layout. If you do not find the settings field in the dialog, please contact your PLC manfacturer. (Please regard: If retain variables are used in an function block instance, the complete instance POU will be stored in the retain memory area !)

3803

"Out of global data memory. Variable '<name>', ‚<number>' bytes."
The memory space available for global variables is exhausted. The size of the memory area can be set target-specific in the target settings /memory layout. If you do not find the settings field in the dialog, please contact your PLC manfacturer.

3820

"'VAR_OUTPUT' and 'VAR_IN_OUT' not allowed in functions"
In a function no output or in_output variables may be defined.

3821

"At least one input required for functions"
Add at least on input parameter for the function.

3840

"Unknown global variable '<name>'!"
In the POU a VAR_EXTERNAL variable is used, for which no global variable declared.

3841

"Declaration of '<name>' do not match global declaration!"
The type given in the declaration of the VAR_EXTERNAL variable is not the same as that in the global declaration.

3900

"Multiple underlines in indentifier"
Remove multiple underlines in the identifier name.

3901

"At most 4 numerical fields allowed in addresses"
There is a direct assignment to an address which has more than four levels. (e.g. %QB0.1.1.0.1).

3902

"Keywords must be uppercase"
Use capital letters for the keyword or activate option ‚Autoformat' in ‚Project' ‚Options'.

3903

"Invalid duration constant"
The notation of the constant does not comply with the IEC61131-3 format.

3904

"Overflow in duration constant"
The value used for the time constant cannot be represented in the internal format. The maximum value which is representable is t#49d17h2m47s295ms.

3905

"Invalid date constant"

The notation of the constant dies not comply with the IEC61131-3 format.

3906

"Invalid time of day constant"
The notation of the constant dies not comply with the IEC61131-3 format.

3907

"Invalid date and time constant"
The notation of the constant dies not comply with the IEC61131-3 format.

3908

"Invalid string constant"
The string constant contains an invalid character.

4000

"Identifier expected"
Enter a valid identifier at this position.

4001

"Variable '<name>' not declared"
Declare variable local or global.

4010

"Type mismatch: Cannot convert '<name>' to '<name>'."
Check what data type the operator expects (Browse Online Help for name of operator) and change the type of the variable which has caused the error, or select another variable.

4011

"Type mismatch in parameter '<name>' of '<name>': Cannot convert '<name>' to '<name>'."
The data type of the actual parameter cannot be automatically converted to that of the formal parameter. Use a type conversion or use another variable type.

4012

"Type mismatch in parameter '<name>' of '<name>': Cannot convert '<name>' to '<name>'."
A value with the invalid type <Typ2> is assigned to the input variable '<name>'. Replace the variable or constant to one of type <Typ1> or use a type conversion respectively a constant with type-prefix..

4013

"Type mismatch in output '<name>' of '<name>': Cannot convert '<name>' to '<name>'."
A value with the invalid type <Typ2> is assigned to the output variable '<name>'. Replace the variable or constant to one of type <Typ1> or use a type conversion respectively a constant with type-prefix..

4014

"Typed literal: Cannot convert '<name>' to '<name>'"
The type of the constant is not compatible with the type of the prefix.
Example: SINT#255

4015

"Data type ‘<name>' illegal for direct bit access"
Direct bit addressing is only allowed for Integer- and Bitstring datatypes. You are using a variable var1 of type Typ REAL/LREAL or a constant in bit access <var1>.<bit>.

4016

"Bit index '<number>' out of range for variable of type '<name>'"
You are trying to access a bit which is not defined for the data type of the variable.

4017

"'MOD' is not defined for 'REAL'"

The operator MOD can only be used for integer and bitstring data types.

4020

"Variable with write access or direct address required for 'ST', 'STN', 'S', 'R'"
Replace the first operand by a variable with write access.

4021

"No write access to variable '%s' allowed"
Replace the variable by a variable with write access.

4022

"Operand expected"
Add an operand behind the command.

4023

"Number expected after '+' or '-'"
Enter a digit.

4024

"<operator 0> or <operator 1> or ... expected before '<name>'"
Enter a valid operand at the named position.

4025

"':=' or '=>' expected before '<name>'"
Enter one of the both operators at the named position.

4026

"'BITADR' expects a bit address or a variable on a bit address"
Use a valid bit address (e.g. %IX0.1).

4027

"Integer number or symbolic constant expected"
Enter a integer number or the identifier of a valid constant.

4028

"'INI' operator needs function block instance or data unit type instance"
Check the data type of the variable, for which the INI operator is used.

4029

"Nested calls of the same function are not possible."
At not reentrant target systems and in simulation mode a function call may not contain a call of itself as a parameter.
Example: fun1(a,fun1(b,c,d),e);
Use a intermediate table.

4030

"Expressions and constants are not allowed as operands of 'ADR'"
Replace the constant or the expression by a variable or a direct address.

4031

"'ADR' is not allowed on bits! Use 'BITADR' instead."
Use BITADR. Please note: The BITADR function does not return a physical memory address

4032

"'<number>' operands are too few for '<name>'. At least '<number>' are needed"
Check how many operands the named operator requires and add the missing operands.

4033

"'<number>' operands are too many for '<name>'. At least '<number>' are needed"
Check how many operands the named operator requires and remove the surplus operands.

4034

"Division by 0"
You are using a division by 0 in a constant expression. If you want to provocate a runtime error, use – if applicable - a variable with the value 0.

4035

"ADR must not be applied on 'VAR CONSTANT' if 'replaced constants' is activated"
An address access on constants for which the direct values are used, is not possible. If applicable, deactivate the option ‚Replace Constants' in ‚Project' ‚Options' ‚Build'.

4040

"Label '<name>' is not defined"
Define a label with the name <LabelName> or change the name <LabelName> to that of a defined label.

4041

"Duplicate definition of label '<name>'"
The label '<name>' is multiple defined in the POU. Rename the label or remove one of the definitions.

4042

"No more than '<number>' labels in sequence are allowed"
The number of jump labels is limited to '<Anzahl>'. Insert a dummy instruction.

4043

"Format of label invalid. A label must be a name optionally followed by a colon."
The label name is not valid or the colon is missing in the definition.

4050

"POU '<name>' is not defined"
Define a POU with the name '<name>' using the command **'Project' 'Add Object'** or change '<name>' to the name of a defined POU.

4051

"'<name>' is no function"
Use instead of <name> a function name which is defined in the project or in the libraries.

4052

"'%s' must be a declared instance of FB '<name>'"
Use an instance of data type '<name>' which is defined in the project or change the type of <Instance name> to '<name>' .

4053

"'<name>' is no valid box or operator"
Replace '<name>' by the name of a POU or an operator defined in the project.

4054

"POU name expected as parameter of 'INDEXOF'"
The given paramter is not a valid POU name.

4060

"'VAR_IN_OUT" parameter '<name>' of '<name>' needs variable with write access as input"
To VAR_IN_OUT parameters variables with write access have to be handed over, because a VAR_IN_OUT can be modified within the POU.

4061

"'VAR_IN_OUT' parameter '<name>' of '<name>' must be used."
A VAR_IN_OUT parameter must get handed over a variable with write access, because a VAR_IN_OUT can be modified within the POU

4062

"No external access to 'VAR_IN_OUT' parameter '<name>' of '<name>'."
VAR_IN_OUT Parameter only may be written or read within the POU, because they are handed over by reference.

4063

"'VAR_IN_OUT' parameter '<name>' of '<name>' must not be used with bit addresses."
A bit address is not a valid physical address. Hand over a variable or a direct non-bit address.

4064

"'VAR_IN_OUT' must not be overwritten in local action call!"
Delete the parameters set for the VAR_IN_OUT variable in the local action call.

4070

"The POU contains a too complex expression"
Decrease nesting depth by dividing up the expression into several expressions. Use intermediate variables for this purpose.

4071

"Network too complex"
Divide up the network into several networks.

4100

"'^' needs a pointer type"
You are trying to dereference a variable which is not declared as a pointer.

4110

"'[<index>]' needs array variable"
[<index>] is used for a variable which is not declared as an array with ARRAY OF.

4111

"Index expression of an array must be of type 'INT'"
Use an expression of the correct type or a type conversion.

4112

"Too many indexes for array"
Check the number of indices (1, 2, oder 3), for which the array is declared and remove the surplus.

4113

"Too few indexes for array"
Check the number of indices (1, 2, oder 3), for which the array is declared and add the missing ones.

4114

"One of the constant indizes is not within the array range"
Make sure that the used indices are within the bounds of the array.

4120

"'.' needs structure variable""
The identifier on the left hand of the dot must be a variable of type STRUCT or FUNCTION_BLOCK or the name of a FUNCTION or a PROGRAM.

4121

" '<name>' is not a component of <object name>"
The component '<name>' is not included in the definition of the object <object name>.

4122

"'%s' is not an input variable of the called function block"
Check the input variables of the called function block and change '<name>' to one of these.

4200

"'LD' expected"
Insert at least one LD instruction after the jump label in the IL editor.

4201

"IL Operator expected"
Each IL instruction must start with an operator or a jump label.

4202

"Unexpected end of text in brackets"
Insert a closing bracket after the text.

4203

"<name> in brackets not allowed"
The operator <name> is not valid in a IL bracket expression.
(not valid are: 'JMP', 'RET', 'CAL', 'LDN', 'LD', 'TIME')

4204

"Closing bracket with no corresponding opening bracket"
Insert an opening bracket or remove the closing one.

4205

"No comma allowed after ')'"
Remove comma after closing bracket.

4206

"Label in brackets not allowed"
Shift jump label so that it is outside of the brackets.

4207

"'N' modifier requires operand of type 'BOOL','BYTE','WORD' or 'DWORD'"
The N modifier requires a data type, for which a boolean negation can be executed.

4208

"Conditional Operator requires type 'BOOL'"
Make sure that the expression gives out a boolean result or use a type conversion.

4209

"Function name not allowed here"
Replace the function call by a variable or a constant.

4210

"'CAL', 'CALC' and 'CALN' require a function block instance as operand"
Declare an instance of the function block which you want to call.

4211

"Comments are only allowed at the end of line in IL"
Shift the comment to the end of the line or to an extra line.

4212

"Accumulator is invalid before conditional statement"
The accu is not defined. This happens if an instruction is preceeding which does not submit a result (e.g. 'CAL').

4213

"'S' and 'R' require 'BOOL' operand"
Use a boolean variable at this place.

4250

"Another 'ST' statement or end of POU expected"
The line does not start with a valid ST instruction.

4251

"Too many parameters in function '%s'"
There are more parameters given than are declared in the definition of the function.

4252

"Too few parameters in function '%s'"
There are less parameters given than are declared in the definition of the function.

4253

"'IF' or 'ELSIF' require 'BOOL' expression as condition"
Make sure that the condition for IF or ELSIF is a boolean expression.

4254

"'WHILE' requires 'BOOL' expression as condition"
Make sure that the condition following the ‘WHILE’ is a boolean expression.

4255

"'UNTIL' requires 'BOOL' expression as condition"
Make sure that the condition following the ‘UNTIL’ is a boolean expression.

4256

"'NOT' requires 'BOOL' operand"
Make sure that the condition following the ‘NOT’ is a boolean expression.

4257

"Variable of 'FOR' statement must be of type 'INT'"
Make sure that the counter variable is of an integer or bitstring data type (e.g. DINT, DWORD).

4258

"Expression in 'FOR' statement is no variable with write access"
Replace the counter variable by a variable with write access.

4259

"Start value in 'FOR' statement is no variable with write access"
The start value in the ‚FOR' instruction must be compatible to the type of the counter variable.

4260

"End value of 'FOR' statement must be of type 'INT'"
The end value in the ‚FOR' instruction must be compatible to the type of the counter variable.

4261

"Increment value of 'FOR' statement must be of type 'INT'"
The incremental value in the ‚FOR' instruction must be compatible to the type of the counter variable.

4262

"'EXIT' outside a loop"
Use 'EXIT' only within 'FOR', 'WHILE' or 'UNTIL' instructions.

4263

"Expecting Number, 'ELSE' or 'END_CASE'"

Within a ‘CASE' expression you only can use a number or a 'ELSE' instruction or the ending instruction 'END_CASE'.

4264

"'CASE' requires selector of an integer type"
Make sure that the selector is of an integer or bitstring data type (e.g. DINT, DWORD).

4265

"Number expected after ','"
In the enumeration of the CASE selectors there must be inserted a further selector after a comma.

4266

"At least one statement is required"
Insert an instruction, at least a semicolon.

4267

"Function block call requires function block instance"
The identifier in the functionblock call is no instance. Declare an instance of the desired functionblock or use the name of an already defined instance.

4268

"Expression expected"
At this position you must enter an expression.

4269

"'END_CASE' expected after 'ELSE'-branch"
Terminate the 'CASE' instruction after the 'ELSE' part with an 'END_CASE'.

4270

"'CASE' constant '<name>' already used"
A 'CASE' selector may only be used once within a ‘CASE' instruction.

4271

"The lower border of the range is greater than the upper border."
Modify the area bounds for the selectors so that the lower border is not highte than the upper border.

4272

"Exptecting parameter '<name>' at place <name> in call of '<name>'!"
You can edit a function call in that way, that also the parameter names are contained, not only the parameter values. But nevertheless the position (sequence) of the parameters must be the same as in the function definition.

4273

Parts of the 'CASE'-Range '<range>' already used in Range '<range>'
Make sure that the areas for the selectors which are used in the CASE instruction, don’t overlap.

4274

"Multiple 'ELSE' branch in 'CASE' statement"
A CASE instruction may not contain more than one ‚ELSE' instruction.

4300

"Jump requires 'BOOL' as input type"
Make sure that the input for the jump respectively the RETURN instruction is a boolean expression.

4301

"POU '%s' need exactly %d inputs"
The number of inputs does not correspond to the number of VAR_INPUT and VAR_IN_OUT variables which is given in the POU definition.

4302

"POU '<name>' need exactly <number> outputs"
The number of outputs does not correspond to the number of VAR_OUTPUT variables which is given in the POU definition..

4303

"<name>' is no operator"
Replace '<name>' by a valid operator.

4320

"Non-boolean expression '<name>' used with contact"
The switch signal for a contact must be a boolean expression.

4321

"Non-boolean expression '<name>' used with coil"
The output variable of a coil must be of type BOOL.

4330

"Expression expected at input 'EN' of the box '<name>' "
Assign an input or an expression to the input EN of POU '<name>'.

4331

"Expression expected at input '<number>' of the box '<name>' "
The input <number> of the operator POU is not assigned.

4332

Expression expected at input '<name>' of the box '<name>'"
The input of the POU is of type VAR_IN_OUT and is not assigned.

4333

"Identifier in jump expected"
The given jump mark is not a valid identifier.

4334

"Expression expected at the input of jump"
Assign a boolean expression to the input of the jump. If this is TRUE, the jump will be executed.

4335

"Expression expected at the input of the return"
Assign a boolean expression to the input of the RETURN instruction. If this is TRUE, the jump will be executed.

4336

"Expression expected at the input of the output"
Assign a suitable expression to the output box.

4337

"Identifier for input expected"
Insert a valid expression or identifier in the input box.

4338

"Box '<name>' has no inputs"
To none of the inputs of the operator POU '<name>' a valid expression is assigned.

4339

"Typemismatch at output: Cannot convert '<name>' to '<name>'.
The type of the expression in the output box is not compatible to that of the expression which should be assigned to it.

4340

"Jump requires 'BOOL' as input type"
Make sure that the input for the jump is a boolean expression.

4341

"Return requires a boolean input"
Make sure that the input for the RETURN instruction is a boolean expression.

4342

"Expression expected at input 'EN' of the box '<name>'"
Assign a valid boolean expression to the EN input of the box.

4343

"Values of Constants: ‘<name>’"
Input '<name>' of box '<name>' is declared as VAR_INPUT CONSTANT. But to this POU box an expression has been assigned in the dialog 'Edit Parameters' which is not type compatible.

4344

"'S' and 'R' require 'BOOL' operand"
Insert a valid boolean expression after the Set resp. Reset instruction.

4345

"Invalid Type for parameter '<name>' of '<name>': Cannot convert '<type>' to '<type>'."
An expression is assigned to input '<name>' of POU box '<name>' which is not type compatible.

4346

"Not allowed to use a constant as an output"
You can only assign an output to a variable or a direct address with write access.

4347

"'VAR_IN_OUT' parameter needs variable with write access as input"
To VAR_IN_OUT parameters only variables with write access can be handed over, because these can be modified within the POU.

4350

"An SFC-Action can not be accessed from outside!"
SFC actions only can be called within the SFC POU in which they are defined.

4351

"Step name is no identifier: '<name>'"
Rename the step or choose a valid identifier as step name.

4352

"Extra characters following valid step name:'<name>'"
Remove the not valid characters in the step name.

4353

"Step name duplicated: '<name>'"
Rename one of the steps.

4354

"Jump to undefined Step: '<name>'"
Choose an existent step name as aim of the jump resp. insert a step with name ‚<name>’.

4355

"A transition must not have any side effects (Assignments, FB-Calls etc.)"
A transition must be a boolean expression.

4356

"Jump without valid Step Name: '<name>' "
Use a valid identifier as aim (mark) of the jump.

4357

"IEC-Library not found"
Check whether the library iecsfc.lib is inserted in the library manager and whether the library paths defined in 'Project' 'Options' 'Paths'are correct.

4358

"Action not declared: '<name>'"
Make sure that in the object organizer the action of the IEC step is inserted below the SFC POU and that in the editor the action name is inserted in the box on the right hand of the qualifier.

4359

"Invalid Qualifier: '<name>'"
In the box on the left hand of the action name enter a qualifier for the IEC action.

4360

"Time Constant expected after qualifier '<name>'"
Enter next to the box on the left hand of the action name a time constant behind the qualifier.

4361

"'<name>' is not the name of an action"
Enter next to the box on the right hand of the qualifier the name of an action or the name of a variable which is defined in the project.

4362

"Nonboolean expression used in action: '<name>'"
Insert a boolean variable or a valid action name.

4363

"IEC-Step name already used for variable: '<name>'"
Please rename the step or the variable.

4364

"A transition must be a boolean expression"
The result of the transition expression must be of type BOOL.

4365

"Time Constant expected after qualifier '<name>'"
Open dialog ,step attributes' for the step '<name>' and enter a valid time variable or time constant

4366

"The label of the parallel branch is no valid identifier: '<name>'"
Enter a valid identifier next to the triangle which marks the jump label.

4367

"The label '<name>' is already used"
There is already a jump label or a step with this name. Please rename correspondingly.

4368

"Action '<name>' is used in multiple step chains, where one is containing the other!"
The action '<name>' is used in the POU as well as in one or several actions of the POU.

4369

"Exactly one network requried for a transition"
There are used several FBD resp. LD networks for a transition. Please reduce to 1 network.

4370

"Additional lines found after correct IL-transition"
Remove the not needed lines at the end of the transition.

4371

"Invalid characters following valid expression: '<name>"
Remove the not needed characters at the end of the transition.

4372

"Step '<name>': Time limit needs type 'TIME'"
Define the time limits of the step in the step attributes by using a variable of type TIME or by a time definition in correct format (e.g "t#200ms").

4373

"IEC-actions are only allowed with SFC-POUs"
There is an action assigned to a non-SFC-POU (see in the Object Organizer), which is programmed in SFC and which contains IEC actions. Replace this action by one which contains no IEC actions.

4374

"Step expected instead of transition '<name>'"
The SFC POU is corrupt, possibly due to any export-import actions.

4375

Import / conversion of POU '<name>' contains errors resp. is not complete."
The SFC POU is corrupt, possibly due to any export-import actions.

4376

"Step expected after transition '<name>'
The SFC POU is corrupt, possibly due to any export-import actions.

4377

"Transition expected after step '<name>'"
The SFC POU is corrupt, possibly due to any export-import actions.

4400

Import / conversion of POU '<name>' contains errors resp. is not complete."
The POU cannot be converted to IEC 61131-3 completely.

4401

"S5 time constant %lu seconds is too big (max. 9990s)."
There is no valid BCD coded time in the accu.

4402

"Direkter Zugriff nur auf E/As erlaubt."
Make sure that you only access variables which are defined as input or output.

4403

"STEP5/7 instruction invalid or not convertible to IEC 61131-3."
Some STEP5/7 commands are not convertable ato IEC 61131-3, e.g. CPU commands like MAS.

4404

"STEP5/7 operand invalid or not convertible to IEC 61131-3."
Some STEP5/7 operands are not convertable ato IEC 61131-3 respectively an operand is missing.

4405

"Reset of a STEP5/7 timer cannot be converted into IEC 61131-3."
The corresponding IEC timer have no reset input.

4406

"STEP5/7 Counter constant out of range (max. 999)."
There is no valid BCD coded ckounter constant in the accu.

4407

"STEP5 instruction not convertible to IEC 61131-3."
Some STEP5/7 instructions cannot be converted to IEC 61131-3, e.g. DUF.

4408

"Bit access of timer or counter words not convertible into IEC 61131-3."
Special timer/counter commands are not convertable into IEC 61131-3.

4409

"Contents of ACCU1 or ACCU2 undefined, not convertible into IEC 61131-3."
A command, which connects the both accus, cannot be converted, because the accu values are not defined.

4410

"Called POU not in project."
Import the called POU.

4411

"Error in global variable list."
Please check the SEQ file.

4412

"Internal error no.11"
Please contact the PLC manufacturer.

4413

"Error in format of line in data block"
In the code which should be imported there is an errouneous date.

4414

"FB/FX name missing."
In the original S5D file the symbolic name of an (extended) POU is missing.

4415

"Instruction after block end not allowed."
A protected POU cannot get imported.

4416

"Invalid Command"
The S5/S7 command´cannot be disassembled.

4417

"Comment not closed"
Close the comment with "*)".

4418

"FB/FX-Name too long (max. 8 characters)"
The symbolic name of an (extended) POU is to long.

4419

"Expected format of line ""(* Name: <FB/FX-Name> *)"" "
Correct the line correspondingly.

4420

"Name of FB/FX parameter missing"
Check the POUs.

4421

"Type of FB/FX parameter invalid"
Check the POUs.

4422

"Type of FB/FX parameter missing"
Check the POUs.

4423

"Invalid FB/FX call parameter"
Check the interface of the POU.

4424

"Warning: FB/FX for call either missing or parameters invalid or has '0' parameters"
The called POU is not imported yet or is not correct or has no parameters (in the last case you can ignore the error message).

4425

"Definition of label missing"
The aim (label) of the jump is not defined.

4426

"POU does not have a valid STEP 5 block name, e.g. PB10"
Modify the POU name.

4427

"Timer type not declared"
Add a declaration of the timer in the global variables list.

4428

"Maximum number of open STEP5 brackets exceeded"
You may not use more than seven open brackets.

4429

"Error in name of formal parameter"
The parameter name may not exceed four characters.

4430

"Type of formal parameter not IEC-convertible"
In IEC 61131-3 Timer, counter and POUs cannot be converted as formal parameters.

4431

"Too many 'VAR_OUTPUT' parameters for a call in STEP5 STL"
A POU may not contain more than 16 formal parameters as outputs.

4432

"Labels within an expression are not allowed"
In IEC 61131-3 jump labels may not be inserted at any desired position.

4434

"Too many labels"
A POU may not contain more than 100 labels.

4435

"After jump / call, a new expression must start"
After jump or call a Load command LD must follow.

4436

"Bit result undefined, not convertible to IEC 61131-3."
The command which is used by VKE verwendet cannot get converted, because the value of the VKE is not known.

4437

"Type of instruction and operand are not compatible"
A bit command is used for a word operand or the other way round.

4438

"No data block opened (insert instruction C DB before)"
Insert a „A DB“.

4500

"Unrecognized variable or address"
The watch variable is not declared within the project. By pressing <F2> you get the input assistant which lists the declared variables.

4501

"Extra characters following valid watch expression"
Remove the surplus signs.

4520

"Error in Pragma: Flag expected before '<name>'!"
The pragma is not correct. Check whether ‘<name>' is a valid flag.

4521

"Error in Pragma: Unexpected element '<name>'!"
Check whether pragma is composed correctly.

4522

"flag off' pragma expected!"
The pragma has not been switched off, insert a 'flag off' command.

4550

"Index out of defined range : Variable OD "number>, Line <line number>."
Ensure that the index is within the area which is defined in the target settings/networkfunctionality.

4551

"Subindex out of defined range : Variable OD "number>, Line <line number>."
Ensure that the subindex is within the area which is defined in the target settings/networkfunctionality.

4552

"Index out of defined range : Parameter OD "number>, Line <line number>."
Ensure that the index is within the area which is defined in the target settings/networkfunctionality.

4553

"Subindex out of defined range : Parameter OD "number>, Line <line number>."
Ensure that the subindex is within the area which is defined in the target settings/networkfunctionality.

4554

"Variablename invalid: Variable OD <number>, Line <line number>."
Enter a valid project variable in the filed ‚variable’. Use the syntax <POU name>.<variable name> resp. for global variables .<variable name>

4555

"Empty table-entry, input not optional: Parameter OD <number>, Line <line number>
You must make an entry in this field.

4556

"Empty table-entry, input not optional: Variable OD <number>, Line <number>"
You must make an entry in this field.

4600

"Networkvariables: '<name>' expression is not from type bool!"
Make sure that the variable defined in the properties dialog of the network variables list at option 'Transmit on event', is of type BOOL.

4900

"Invalid type for conversion"
You are using a type conversion which is not supported by the currently chosen codegenerator.

12 Index

A

ABS 11-34
Absolute Value 11-34
Access rights 4-55
ACOS 11-37
Action 2-7, 2-18, 2-19, 3-7, 4-64
Action Init 3-6
Active step 2-20
ADD 11-13
Address Function 11-27
Addresses 11-66
ADR 11-27
ALIAS 11-10
Alternative Branch in SFC 2-24, 5-36
AND 11-17
Application-specific parameter dialog 6-27
Arc cosine 11-37
Arc sine 10-26 11-37
Arc tangent 11-37
Argument 2-2, 2-4
ARRAY 11-6
ASIN 11-37
Assignment 2-12
Assignment Combs 5-26
Assignment operator 2-14
AT Declaration 5-6
ATAN 11-37
Auto Load 4-5
Auto Save 4-5
Autodeclaration 4-7, 5-8
Autoformat 4-7

B

Backup, automatic 4-5
Basic parameters
  DP slave 6-36
Basisparameter
  Bitchannel 6-32
  Channel 6-31
  DP master 6-33
  I/O Module 6-28
Batch commands 11-69
BCD conversion 11-53
BCD_TO_INT 11-53
Binary file of the application 4-13
binary symbol information 4-21
Binding of ST operators 2-12
Bistable Function Blocks 11-43
Bit addressing 11-66
Bit-addressed variables 5-22
BITADR 11-27
Bitchannel 6-32
Bitmap 7-4, 7-28
Bitvalues 4-8
BLINK 11-57
Body 5-1, 5-18, 5-19, 5-28
Bookmark in Help 4-95
BOOL 11-5
BOOL Constants 11-63
BOOL_TO Conversions 11-28
Breakpoint 1-2, 2-28, 5-15, 5-17
Breakpoint Dialog Box 4-77
Breakpoint position 4-76
Browser ini-file 6-16
Build 4-12, 4-32
Bus parameters
  DP master 6-34
BYTE 11-5
BYTE Constants 11-64

C

CAL 11-28
CALC 2-10
Call tree 4-32, 4-65
Calling a function 2-2
Calling a function block 2-4, 2-13
Calling function blocks in ST 2-14
Calling POUs 5-15
Callstack in Taskconfiguration 6-54
CAN Configuration 6-41
CAN Master configuration 6-43
CAN Modular Slaves 6-45
CAN Module configuration 6-43, 6-45
CASE 2-13
CASE instruction 2-15
CFC 2-25
Changing connections in CFC 5-50
Channelparameter 6-32
CHARCURVE 11-59
Check 4-32, 4-48
Check in 4-20
Check out 4-20
CheckBounds 2-2
CheckDivByte 2-2
CheckDivDWord 2-2
CheckDivReal 2-2
CheckDivWord 2-2
CheckRangeSigned 2-2, 11-11
CheckRangeUnsigned 2-2, 11-11
COB-Id 6-42
CoDeSys 1-1
CoDeSys operating version 7-26
Coil 2-26, 5-31
Collapse Node 4-51
Colors 4-9, 7-16
Command file 11-69
Command Line 11-69
Comment 5-10, 5-21
Communication
  Symbol interface 4-15
Communications Parameters Gateway 4-84

Comparing projects 4-41
Compile File options 4-21
compile files in data base 4-17
Compress 6-59
CONCAT 11-41
Concatenation 11-41
Concurrent Access 4-32
Configure element 7-11
Connection marker in CFC 5-51
CONSTANT 5-5
Constants 5-5
Contact 2-26, 5-29
Content Operator 11-8, 11-28
Context menu 4-4
Context Sensitive Help 4-96
Continuous function chart editor 2-25
Controller 4-74
Conversion of Integral Number Types 11-31
Conversions of types 11-28
Convert object 4-53
Converting S5 to IEC 1131-3 11-78
Copy 4-67
Copy in Help 4-94
Copying in CFC 5-49
Copying in FBD 5-27
COS 11-36
Cosine 11-36
Counter 11-46
Create Backup 4-5
Create binary symbol information 4-21
Create boot project 4-21
Create bootproject 4-89
Creating connections in CFC 5-49
Cross reference list 4-65
Cross-reference list 4-32
CTD 11-47
CTU 11-46
CTUD 11-47
Custom Parameters 6-32
Cut 4-67
Cutting in FBD 5-27

D

Data base commands 4-56
  Add Shared Objects 4-62
  Check In 4-58
  Check Out 4-58
  Define 4-58
  Get All Latest Versions 4-60
  Get Latest Version 4-58
  Label Version 4-61
  Login 4-63
  Multiple Check In 4-61
  Multiple Check Out 4-61
  Multiple Define 4-60
  Multiple Undo Check Out 4-61
  Project Version History 4-61
  Refresh Status 4-63
  Show Differences 4-59
  Show Version History 4-59
  Undo Check Out 4-58
Data base functions 4-17
Data Base Link 4-17
data base object category 4-17
Data types 2-9, 4-2, 11-5
database project 4-19
DATE 11-6
DATE Constants 11-63
DATE_AND_TIME 11-6
DATE_AND_TIME Constants 11-64
DATE_TO Conversions 11-32
DCF file 6-4
DDE Interface 8-1
Debugger 5-15
Debugging 2-28, 4-12, 5-21
Declaration 3-4, 5-2
  Array 5-9
  Field 5-9
  flag 5-11
Declaration Editor 5-2
Declaration Part 2-1, 5-1, 5-2, 5-18, 5-19, 5-28
Declarations as tables 4-7, 5-10
Declare Variable 4-73
Declare, automatic 4-7, 5-8
Decrementer 11-47
Defined Data Types 11-6
Delete 4-68
DELETE 11-41
Deleting a Transition 5-38
Deleting an Action 5-38
Deleting connections in CFC 5-50
Deleting in FBD 5-27
Dereferencing 11-8, 11-28
DERIVATIVE 11-55
Desktop 4-8
DINT 11-5
DINT Constants 11-64
Directory 4-10
DIV 11-14
Document 4-30, 4-38
Document Frame 6-9, 6-15
Download 4-14, 4-75, 4-89
Download-Information 4-33
DP parameters
  DP master 6-33
  DP slave 6-37
Drag&Drop 4-50, 4-51
DT 11-6
DT_TO Conversions 11-32
DWORD 11-5
DWORD Constants 11-64

E

Edit Menu
  Copy 4-67
  Cut 4-67, 5-27
  Delete 4-68
  Find 4-69

Find next 4-69
Input Assistant 4-70
Next error 4-73
Paste 4-68, 5-27
Previous error 4-73
Redo 4-66
Replace 4-70
Undo 4-66
Editing functions 4-66
Editor options 4-6
Editors 5-1, 5-18, 5-19, 5-28
Einfügen in KOP 5-33
Emergency Telegram 6-44
EN Input 2-27, 5-31
EN POU 2-27
END_PROGRAM 2-6
END_TYPE 11-8, 11-9, 11-10
Engineering Interface 9-7
ENI 9-7
Open file from ENI 4-24
ENI Admin 9-8
ENI Control 9-8
ENI object categories 9-9
ENI Server 9-7
ENI Server Suite 9-8
Entering Trace Variables 6-55
Entry action 5-37
Entry action 2-20
Enumeration 11-8
EQ 11-26
EXIT 2-13, 2-18
Exit action 5-37
Exit action 2-20
EXP 11-35
Expand Node 4-51
Exponential Function 11-35
Exponentiation 11-38
Export 4-39
export file 6-4
Expression 2-12
EXPT 11-38
EXTERNAL 5-5
External library 4-26
External Library 6-11
External variables 5-5
EXTRACT 11-54
Extras Menu
Align 7-6, 7-37
Associate Action 5-41
Auto Read 6-56
Clear Action/Transition 5-38
Clear Background Bitmap 8-12 7-34
Compress 6-59
Configure 7-11
Cursor Mode 6-58
6-54
Element list 7-7
Keyboard Usage 7-34
Link Docuframe File 6-10, 6-15
Load Trace 6-59
Load Watch List 6-61
Make Docuframe File 6-10, 6-15
Monitoring Active 6-62
Multi Channel 6-58
Options 5-40
Paste after 5-38
Read Receipt 6-63
Read Trace 6-56
Rename Watch List 6-61
Save Trace 6-59
Save Watch List 6-61
7-37
Select Background Bitmap 7-34
Send to Back 7-6, 7-34
Send to Front 7-6, 7-34
6-54
Set/Reset 5-26
Settings 7-31
Show grid 6-58
Start Trace 6-56
Step Attributes 5-38
Stop Trace 6-57
Stretch 6-59
Time Overview 5-39
Trace Configuration 6-55
Trace in ASCII-file 6-59
Use IEC Steps 5-41
Write Receipt 6-62
Y Scaling 6-58
Zoom 5-1
Zoom Action/Transition 5-38

'

'Extras' 'Accept access rights 4-45
'Extras' 'Accept changed item' 4-45
'Extras' 'Äccept change' 4-44
'Extras' 'Accept properties' 4-45

E

Extras Menu
Callstack 6-54
Monitoring Options 5-16
Negate 5-26
Options 5-21
Properties 6-12
Set Debug Task 6-54

'

'Extras' 'Next difference' 4-44
'Extras' 'Previous difference' 4-44

F

F_TRIG 11-45
F4 4-9
falling edge 11-45

FBD 2-2, 2-5, 2-24, 4-76, 5-22
FBD Editor 5-22
Feedback paths in CFC 5-56
Fields 2-1, 11-6
File 4-23
File Menu
  Close 4-25
  Exit 4-32
  New 4-23
  Open 4-24
  Print 4-29
  Printer Setup 4-30
  Save 4-25
  Save as 4-25
  Save/Mail Archive 4-26
FIND 11-42
Find' 4-69
Flag
  noinit 5-11
  noread 5-11
  nowatch 5-11
  nowrite 5-11
Flow control 5-19
Flow Control
  FBD 5-28
Folder 4-50, 4-51
Font 4-7
FOR 2-16
FOR loop 2-13, 2-16
Force 4-79
Force values 4-79
Forcing 5-14, 6-63
formatting text in visualization 7-13
Formatting, automatic 4-7
Frame 7-32
Function 2-1, 11-67
FUNCTION 2-1
Function Block 2-3
Function block call 2-4
Function Block Diagram 2-2, 2-5, 4-76, 5-22
Function block in LD 2-27
Function block, instance 2-3
Function call 2-2
Function declaration 2-1
FUNCTION_BLOCK 2-3

G

GatewayDDE Server 8-2
GE 11-26
GEN 11-58
Get latest version 4-19
Global constants 5-5
Global Constants 6-7
Global Variable List 6-3
Global variables 6-2
  Network variables 6-6, 6-7
  Persistent variables 6-7
  remanent variables 6-7
  Retain variables 6-7
Global Variables 6-2
Graphic Editors 5-20
Grid 7-32
GT 11-25

H

Heartbeat Master 6-43
Heartbeat Settings 6-44
Help 4-93
Help Menu
  Contents and Index 4-93
Help Topics Window 4-93
Hotkeys in Visualization 7-34
HYSTERESIS 11-61

I

Identifier 5-6, 11-65
IEC 61131-3 2-30
IEC Step 2-20, 5-41
IEC steps 2-21
Iecsfc.lib 2-21
IF instruction 2-13, 2-15
IL 2-2, 2-3, 2-4, 2-9, 4-76, 5-18
IL Editor 5-18
IL operator 2-10
 4-15
Implicit variables in SFC 2-22
Import 4-40
Import from a S5 Project File 11-78
Import from a SEQ Symbol File 4-40, 11-77
Import from S5 File 4-40
Incrementer 11-46
Incrementer/Decrementer 11-47
Index Window 4-95
INDEXOF 11-16
Initialization 5-6
Input and Output Variable 5-3
Input assistant
  structured 4-70
  unstructured 4-70
Input Assistant 4-70
Input in FBD 5-25
INSERT 11-41
Insert in SFC 5-35
Insert Menu
  Add Entry-Action 5-37
  Add Exit-Action 5-37
  Additional Library 6-11
  All Instance Paths 6-9
  Alternative Branch (left) 5-36
  Alternative Branch (right) 5-36
  Append Program Call 6-51
  Append Task 6-49
  Bitmap 7-3, 7-4
  7-3
  Comment 5-21
  Declarations Keywords 5-7

Ellipse 7-3
Function Block 5-15
Function with EN 5-32
Input 5-25
Insert at Blocks 5-31
Insert Program Call 6-51
Insert Task 6-49
Jump 5-36
Line 7-4
Metafile 7-3
Network (after) 5-21
Network (before) 5-21
New Declaration 5-11
New Watch List 6-61
Output 5-26
Parallel Branch (left) 5-36
Parallel Branch (right) 5-36
Parallel Contact 5-30
Placeholder 4-31
Polygon 7-3
Rectangle 7-3
Rounded Rectangle 7-3
Step Transition (after) 5-36
Step Transition (before) 5-29 5-35
Transition-Jump 5-37
Types 5-7
Visualization 7-3, 7-4
Insert mode 5-14
Insert Network 5-21
Insert visualization elements 7-2
Instance 2-3, 5-2
Instance name 2-3, 2-4
Instruction 2-9, 2-12, 2-14
Instruction List 2-2, 2-3, 2-4, 2-9, 4-76, 5-18
INT 11-5
INT Constants 11-64
INT_TO_BCD 11-53
INTEGRAL 11-55
Intellisense functionality 5-2
Internal library 4-26
Internal Library 6-11

J

JMPC 2-10
Jump 2-24
Jump in SFC 5-35, 5-36
Jump Label 5-37, 5-38

K

Keyboard Usage in Visualization 7-37
Keywords 5-5, 5-7
Kommentar 5-1
Kommentare ignorieren 4-42

L

Label 5-20, 5-37
Ladder Diagram 2-25, 4-76, 5-28
Language 7-32
LD 2-25, 4-76, 5-28
LD Editor 5-28
LE 11-25
lecsfc.lib 2-20
Leerzeichen ignorieren 4-42
LEFT 11-39
LEN 11-39
library
license 10-11
Library 2-9, 4-26
Properties 6-12
Library directory 4-10
Library Manager 3-4, 6-10
Library, Define 6-11
Library, Insert 6-11
license information 10-11
Licensing a library 4-26
Licensing Manager 10-11
LIMIT 11-24
LIMITALARM 11-61
Line number field 4-76, 4-83, 5-17
Line Number of the Text Editor 5-17
Line numbers 5-10
Linebreaks in Network Comments 5-21
LN 11-35
Load & Save 4-4
Load file from controller 4-89
Load Trace 6-59
Load trace configuration 6-56
Load Watch List 6-61
local objects 4-17
Local Variable 5-4
Log 4-11, 4-90, 6-12
LOG 11-35
Log in 4-74
Logarithm 11-35
Login to ENI data base 4-19
Logout 4-75
Loop 2-11, 2-14
LREAL 11-5
LREAL Constants 11-64
LREAL_TO Conversions 11-31
LT 11-25

M

Macro 4-13
Define 4-22
Save as library 4-23
Macro after compile 4-13
Macro before compile 4-13
Macro in CFC 5-55
Macro libraries 4-23
Macro options 4-22
Main Help Window 4-94
Main program 2-7
Makros 4-73
Mark 4-8

Marking in SFC 5-35
MAX 11-23
Memory location 11-67
Menu Bar 4-1
Menü Bearbeiten
  Makros 4-73
Menu Bit-addressed variables 5-16
Menü Edit
  Declare Variable 4-73
Menu Extras
  Add label to parallel branch 5-37
  Back all macro level 5-56
  Back one macro level 5-56
  Connection marker 5-51
  Create macro 5-55
  Display order 5-52
  Edit macro 5-56
  EN/ENO 5-47
  Expand macro 5-56
  Insert above 5-33
  Insert below 5-33
  Keyboard Usage 7-37
  Negate 5-47
  Negation 5-33
  Open instance 5-2
  Order - One backwards 5-53
  Order - One forwards 5-53
  Order – To the beginning 5-53
  Order – To the end 5-53
  Order everything according to data flow 5-53
  Order topologically 5-52
  Paste Parallel Branch (right) 5-37
  Properties 5-48, 6-42
  Replace element 6-26
  Select All 5-49
  7-37
  Set/Reset 5-34, 5-47
Menü Extras
  Dahinter Einfügen 5-33
  List of Placeholders 7-9
  Show grid 6-58
Menu Insert
  Append subelement 6-25
  Assignment 5-24
  Box 5-45
  Button 7-4
  Coil 5-31
  Comment 5-46
  Contact 5-29
  Curve 7-4
  In-Pin 5-47
  Input 5-46
  Input of box 5-46
  Insert at POU 5-32
  Insert element 6-25
  Jump 5-24, 5-33, 5-46
  Label 5-46
  Out-Pin 5-47
  Output 5-46
  POU 5-31
  POU with EN 5-32
  Return 5-24, 5-33, 5-46
Menu Online
  Communications Parameters Gateway 4-84
  Create bootproject 4-89
  Force values 4-79
  Load file from controller 4-89
  Release force 4-80
  Reset 4-76
  Sourcecode download 4-89
  Write file to controller 4-89
  Write values 4-78
  Write/Force dialog 4-81
Menu project
  Data Base Link 4-56
  Open instance 4-64
  Properties 4-56
Menu Project
  Add Action 4-64
  Check 4-32
  Clean all 4-33
  Load Download-Information 4-33
  Rebuild all 4-33
  Translate into another language 4-33
Merge 4-45
Message window 4-3, 4-47
MID 11-40
MIN 11-23
MOD 11-15
Modifier 2-10
Modifiers 11-97
Module parameters
  DP slave 6-41
Modulparameter 6-30
  DP master 6-33
Monitoring 2-28, 4-74, 5-13, 5-15, 6-62
Moving in CFC 5-49
MUL 11-14
Multi Channel 6-58
Multiple Write Access 4-32
MUX 11-24

N

NE 11-27
Negation in FBD 5-26
Negation in LD 5-33
Nested comments 4-13
Network 5-20, 5-21, 5-23
Network Comments 5-21
Network in FBD 2-24, 3-2
Network in LD 2-26
Network in SFC 3-6
Network Linebreaks 5-21
Network number 5-20
Network number field 4-76, 4-83
Network variables 6-2
Netzwerkvariablen 6-6
New Folder 4-51
Next error 4-73

Nodeguarding 6-44
NOT 11-18
Note in Help 4-94
Notice at load 4-14
Number Constants 11-64
Number of data 4-13

O

Object 2-1, 4-50
Object add to ENI object category 4-53
object categories in project data base Link 4-17
Object Dicitionary 6-19
Object Organizer 4-2
Object properties 4-56
OD Editor 6-20
Online 1-2, 5-15, 5-21, 6-62
online Menu
  Breakpoint Dialog Box 4-77
  Download 4-75
  Reset 4-75
  Run 4-75
  Show Call Stack 4-82
  Simulation 4-83
  Single Cycle 4-78
  Step in 4-78
  Step over 4-77
  Stop 4-75
  Toggle Breakpoint 4-76
online Mode
  CFC 5-57
  Declaration Editor 5-13
  Function Block Diagram Editor 5-27
  Ladder Editor 5-34
  Network Editor 5-21
  PLC Configuration 6-47
  S FC 5-41
  Text Editor 5-15
  Watch- and Receipt Manager 6-62
Online Change 4-73
Online functions 4-73
Online in Security mode 4-9
Online Menu
  Log in 4-74
  Logout 4-75
Online Mode 4-74
Open file from ENI 4-24
Open instance 4-64
Open POU 4-64
Open project from PLC 4-24
Operand 2-2, 11-63
Operating version 7-26
Operator in FBD 3-4
Operators 11-13, 11-97
  overview 11-97
Oppose differences 4-42
Options 4-4
OR 11-17
Order of execution in CFC 5-51
Output in FBD 5-26
Output Variable 5-3
Overwrite mode 5-14

P

PACK 11-54
Parallel Branch in SFC 2-24, 5-36
Parallel Contacts 2-26, 5-30
Parameter Manager 6-19
Password 4-13
Passwords for user groups 4-49
Pasting 4-68
Pasting in FBD 5-27
PD 11-56
PDO 6-45
PDO mapping of a CAN module 6-45
Persistent global variables 6-7
Persistent variable 5-4
PID 11-57
Placeholder 4-31, 7-30
Placeholders 7-9
PLC Browser 6-15
  Commands 6-16
  Macros 6-17
PLC Configuration in Online Mode 6-47
PLC_PRG 2-7
Pointer 11-8
POINTER 11-8
POU (Program Organization Unit) 1-1, 2-6, 3-1, 4-2
Pragma 5-11
Presentation 7-32
Previous error 4-73
Print 4-29
Print in Help 4-94
Print margins 5-1
Printer borders 4-9
printf 7-13
Profibus Master
  Basisparameter 6-33
  Bus parameters 6-34
  DP parameters 6-33
  Modulparameter 6-33
Profibus Slave
  Basic parameters 6-36
  DP parameters 6-37
  Group assignment 6-40
  In-/outputs 6-38
  Module parameters 6-41
  User parameters 6-40
Program 2-6
PROGRAM 2-6
Program call 2-6
Project 1-1, 2-1, 2-6, 3-1
Project directory 4-10
project info 4-45
Project info 4-5
Project Menu
  Add object 4-52
  Build 4-32
  Check 4-48

Compare 4-41
Convert object 4-53
Copy object 4-54
Delete Object 4-52
Document 4-38
Export 4-39
Global Replace 4-48
Global Search 4-47
Import 4-40
Merge 4-45
4-55
Open object 4-55
Options 4-4
Passwords for user groups 4-49
Project info 4-45
Rename object 4-53
Show call tree 4-65
Show cross reference list 4-65
Siemens Import 4-40
project objects in data base 4-17
Project Objects options 4-18
Project source control
Activate 4-17
Options 4-16
Project version 1.5 4-26
Project version 2.0 4-26
Project version 2.1 4-26
Project version 2.2 4-26
PUTBIT 11-54

Q

Qualifier 2-21

R

R_TRIG 11-45
RAMP_INT 11-60
RAMP_REAL 11-60
Read Receipt 6-63
Read trace 6-56
Read Trace 6-56
REAL 11-5
REAL Constants 11-64
REAL_TO Conversions 11-31
Receipt Manager 6-60
Redo 4-66
References 11-10
Remanent global variables 6-7
Remanent variable 5-4
REPEAT 2-13
REPEAT loop 2-18
Replace 4-48, 4-70
REPLACE 11-42
Replace constants 4-13
Reset 4-75, 4-76
Reset output 5-34
Reset Output 5-26
Resources 2-8, 4-2, 6-1
Retain 2-4
Retain variable 5-4
Retain-Variables 6-7
RETURN 2-13, 2-15
Revision control 9-7
RIGHT 11-40
rising edge 11-45
Rising edge detection 11-45
ROL 11-20
ROR 11-21
Rotation 11-20, 11-21
RS 11-43
RTC 11-51
Run 4-75

S

6-55, 6-56
Sampling Trace 2-27, 6-54
Save 4-25
Save Trace 6-59
Screen divider 4-2
sdb-file 4-21
SDO 6-47
SEL 11-22
Selecting 4-8
Selecting in CFC 5-49
SEMA 11-44
Sequential Function Chart 2-2, 2-5, 2-18, 3-6, 4-76, 5-34
Sequential Function Chart editor
Tooltip 5-42
Tooltips 5-34
Set output 5-34
Set Output 5-26
Set/Reset coils 2-27
SFC 2-2, 2-5, 2-18, 3-6, 4-76, 5-34
Delete Step and Transition 5-36
SFC Editor 5-34
SFC Flags 2-22
SFC library 2-20, 2-21
SFCCurrentStep 2-23
SFCEnableLimit 2-23
SFCError 2-23
SFCErrorAnalyzation 2-23
SFCErrorPOU 2-23
SFCErrorStep 2-23
SFCInit 2-23
SFCPause 2-23
SFCQuitError 2-23
SFCReset 2-23
SFCTip 2-24
SFCTipMode 2-24
SFCTrans 2-23
shared objects in data base 4-17
Shared Objects options 4-20
Shift 11-19
SHL 11-19
Shortcut Mode 5-7
Show Call Stack 4-82
Show grid 6-58

SHR 11-20
Siemens Import 4-40, 11-77
Simulation 2-29, 3-13, 4-74, 4-83
SIN 11-36
Sine 11-36
Single Cycle 4-78
Single step 5-42
Single Step 2-28, 4-77
SINT 11-5
SINT Constants 11-64
SIZEOF 11-16
Sourcecode download 4-89
Sourcedownload 4-14
sprintf 7-13
SQRT 11-34
Square Root 11-34
SR 11-43
ST 2-2, 2-5, 2-11, 4-76, 5-19
ST Editor 5-19
ST operand 2-12
ST operator 2-12
Standard data types 11-5
Standard Function 6-11
Standard Library 3-4, 6-11
Standard POUs 2-1
Standard.lib 6-11
Start Trace 6-56
Statistics 4-46
STATISTICS_INT 11-55
STATISTICS_REAL 11-56
Status bar 4-3, 4-9, 7-8
Step 2-19, 4-76, 5-35
  Delete 5-36
Step Attributes 5-38
Step Init 2-20
Stepping 5-15, 5-22
Stop 4-75
Stop Trace 6-57
Stretch 6-59
STRING 11-6
STRING Constants 11-64
String functions 11-39
STRING_TO Conversions 11-33
STRUCT 11-9
Structured Text 2-2, 2-5, 2-11, 4-76, 5-19
Structures 2-1, 11-9
SUB 11-14
Subrange types 11-10
Symbol configuration 4-15
Symbol file 4-15, 5-11
Symbol interface 4-15
sym-file 4-21
Syntax Coloring 5-2, 5-7
System Events 6-51
System Flag 11-66

T

Tab-width 4-7
TAN 11-36
Tangent 11-36
Target 6-63
Target file 6-64
Target Settings 6-63, 11-85
Target Support Package 6-63
Target-Visualization 7-40
Task Configuration 6-47
Template 5-1
Text editors 5-14
TIME 11-6
TIME Constants 11-63
Time Management in SFC Editor 5-39
TIME_OF_DAY 11-6
TIME_OF_DAY Constants 11-63
TIME_TO Conversions 11-32
TIME-Function 11-67
Timer 11-48
tnf-Datei 6-64
TO_BOOL Conversions 11-30
TOD 11-6
TOD_TO Conversions 11-32
TOF 11-50
TON 11-49
Tool bar 4-2, 4-9
Tooltip 4-2, 4-50, 5-1, 5-16, 5-22, 5-28, 5-34
  Sequential Function Chart editor 5-42
Tooltips
  Sequential Function Chart editor 5-34
TP 3-4, 11-48
Trace Buffer 6-54, 6-57
Trace in ASCII-file 6-59
Trace Variable 6-57
Trace, Automatically Read 6-56
Transition 2-20, 5-35
  Delete 5-36
Translate into another language 4-33
Translation file 4-33, 4-36
  Visualization texts 7-14
Trigger 2-27, 6-55, 11-45, 11-48
 6-55
 6-55
 6-55, 6-56
TRUNC 11-33
TSP 6-63
Turn-off delay 11-50
Turn-on delay 11-49
TYPE 11-8, 11-9, 11-10
Type Conversions 11-28
Typed Literals 11-65
Types 5-7

U

UDINT 11-5
UDINT Constants 11-64
UINT 11-5
UINT Constants 11-64
Undo 4-66
UNPACK 11-54
User group 4-48

User information 4-6
User parameters
  DP slave 6-40
USINT 11-5
USINT Constants 11-64

V

VAR 5-4, 5-8
VAR_CONFIG 6-2, 6-8
VAR_CONSTANT 6-7
VAR_GLOBAL 5-8, 6-2, 6-6
VAR_IN_OUT 5-3
VAR_INOUT 5-8
VAR_INPUT 5-3, 5-8
VAR_INPUT CONSTANT 5-49
VAR_OUTPUT 5-3, 5-8
VAR_RETAIN 6-7
Variable Configuration 6-8
Variable declaration 5-11
Variable name 5-6
Variables 11-65
Variables declaration 5-5
VARIANCE 11-56
Visualisation
  Input possibilities for the operating version 7-26
Visualisierung
  Line width 7-16
  List of Placeholders 7-9
Visualization 2-9, 4-2, 7-2
  Bitmap 7-28
  Colors 7-16
  File - Print 7-38
  in Libraries 7-38
  Input 7-19
  Motion absolute 7-17
  Motion relative 7-18
  Online Mode 7-37
  Operation in online mode 7-37
  Operation over the keyboard 7-37
  Placeholder 7-30
  Shape 7-12
  Text 7-13
  Tooltip 7-28
  Variables 7-19
Visualization Elements, Configure 7-11
Visualization Elements, Shift, Move 7-6
Visualization Object
  Configuration 7-31
Voreinstellung 11-85

W

Watch and Receipt Manager 6-60
Watch and Receipt Manager Offline 6-60
Watch and Receipt Manager Online 6-62
Watch List 6-60
Watch Variable 5-13, 5-28
Web-Visualization 7-38
WHILE loop 2-13, 2-17
Window' 4-92
Window Menu
  Arrange symbols 4-93
  Cascade 4-93
  Close all 4-93
  Library Manager 4-93, 6-10
  Log 4-93
  Messages 4-93
  Tile Horizontal 4-92
  Tile Vertical 4-92
WORD 11-5
WORD Constants 11-64
Work space 4-3
Write 4-78
Write file to controller 4-89
Write protection password 4-14
Write Receipt 6-62
Write values 4-78

X

XOR 11-18

Y

Y Scaling 6-58

Z

Zoom 5-1, 5-20
Zoom Action 5-38
Zoom to POU 5-1
Zoom Transition 5-38